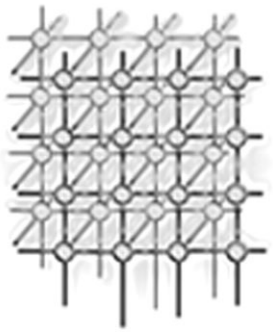
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2002; **14**:1175–1220 (DOI: 10.1002/cpe.710)



GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing

Rajkumar Buyya[1,*,†] and Manzur Murshed[2]

[1]*Grid Computing and Distributed Systems (GRIDS) Lab., Department of Computer Science and Software Engineering, The University of Melbourne, 221 Bouverie St., Carlton, Melbourne, Australia*
[2]*Gippsland School of Computing and IT, Monash University, Gippsland Campus, Churchill, Vic. 3842, Australia*

SUMMARY

Clusters, Grids, and peer-to-peer (P2P) networks have emerged as popular paradigms for next generation parallel and distributed computing. They enable aggregation of distributed resources for solving large-scale problems in science, engineering, and commerce. In Grid and P2P computing environments, the resources are usually geographically distributed in *multiple administrative domains*, managed and owned by different organizations with different policies, and interconnected by wide-area networks or the Internet. This introduces a number of resource management and application scheduling challenges in the domain of security, resource and policy heterogeneity, fault tolerance, continuously changing resource conditions, and politics. The resource management and scheduling systems for Grid computing need to manage resources and application execution depending on either resource consumers' or owners' requirements, and continuously adapt to changes in resource availability.

The management of resources and scheduling of applications in such large-scale distributed systems is a complex undertaking. In order to prove the effectiveness of resource brokers and associated scheduling algorithms, their performance needs to be evaluated under different scenarios such as varying number of resources and users with different requirements. In a Grid environment, it is hard and even impossible to perform scheduler performance evaluation in a *repeatable* and *controllable* manner as resources and users are distributed across multiple organizations with their own policies. To overcome this limitation, we have developed a Java-based discrete-event Grid simulation toolkit called *GridSim*. The toolkit supports modeling and simulation of heterogeneous Grid resources (both time- and space-shared), users and application models. It provides primitives for creation of application tasks, mapping of tasks to resources, and their management. To demonstrate suitability of the GridSim toolkit, we have simulated a Nimrod-G

*Correspondence to: Rajkumar Buyya, Grid Computing and Distributed Systems (GRIDS) Lab., Department of Computer Science and Software Engineering, The University of Melbourne, 221 Bouverie St., Carlton, Melbourne, Australia.
†E-mail: raj@cs.mu.oz.au
Authors listed in order of surname (to signify their similar effort).

Received 6 November 2001
Revised 9 February 2002

like Grid resource broker and evaluated the performance of deadline and budget constrained cost- and time-minimization scheduling algorithms. Copyright © 2002 John Wiley & Sons, Ltd.

KEY WORDS: Grid computing; modelling; simulation; scheduling; performance evaluation

1. INTRODUCTION

The proliferation of the Internet and the availability of powerful computers and high-speed networks as low-cost commodity components are changing the way we do large-scale parallel and distributed computing. The interest in coupling geographically distributed (computational) resources is also growing for solving large-scale problems, leading to what is popularly called the Grid [1] and peer-to-peer (P2P) computing [2] networks. These enable sharing, selection and aggregation of suitable computational and data resources for solving large-scale data intensive problems in science, engineering, and commerce. A generic view of Grid computing environment is shown in Figure 1. The Grid consists of four key layers of components: fabric, core middleware, user-level middleware, and applications [3]. The Grid fabric includes computers (low-end and high-end computers including clusters), networks, scientific instruments, and their resource management systems. The core Grid middleware provides services that are essential for securely accessing remote resources uniformly and transparently. The services they provide include security and access management, remote job submission, storage, and resource information. The user-level middleware provides higher-level tools such as resource brokers, application development and adaptive runtime environment. The Grid applications include those constructed using Grid libraries or legacy applications that can be Grid enabled using user-level middleware tools.

The user essentially interacts with a resource broker that hides the complexities of Grid computing [4,5]. The broker discovers resources that the user can access using information services, negotiates for access costs using trading services, maps tasks to resources (scheduling), stages the application and data for processing (deployment), starts job execution, and finally gathers the results. It is also responsible for monitoring and tracking application execution progress along with adapting to the changes in Grid runtime environment conditions and resource failures.

The computing environments comprise heterogeneous resources (PCs, workstations, clusters, and supercomputers), fabric management systems (single system image OS, queuing systems, etc.) and policies, and applications (scientific, engineering, and commercial) with varied requirements (CPU, input/output (I/O), memory and/or network intensive). The users: *producers* (also called *resource owners*) and *consumers* (also called *end-users*) have different goals, objectives, strategies, and demand patterns. More importantly both resources and end-users are geographically distributed with different time zones. In managing such complex Grid environments, traditional approaches to resource management that attempt to optimize system-wide measures of performance cannot be employed. This is because traditional approaches use centralized policies that need complete state information and a common fabric management policy, or decentralized consensus based policy. In large-scale Grid environments, it is impossible to define an acceptable system-wide performance matrix and common fabric management policy. Apart from the centralized approach, two other approaches that are used in distributed resource management are: *hierarchical* and *decentralized* scheduling or a combination of them [6]. We note that similar heterogeneity and decentralization complexities exist in human economies where market driven economic models have been used to successfully manage them.

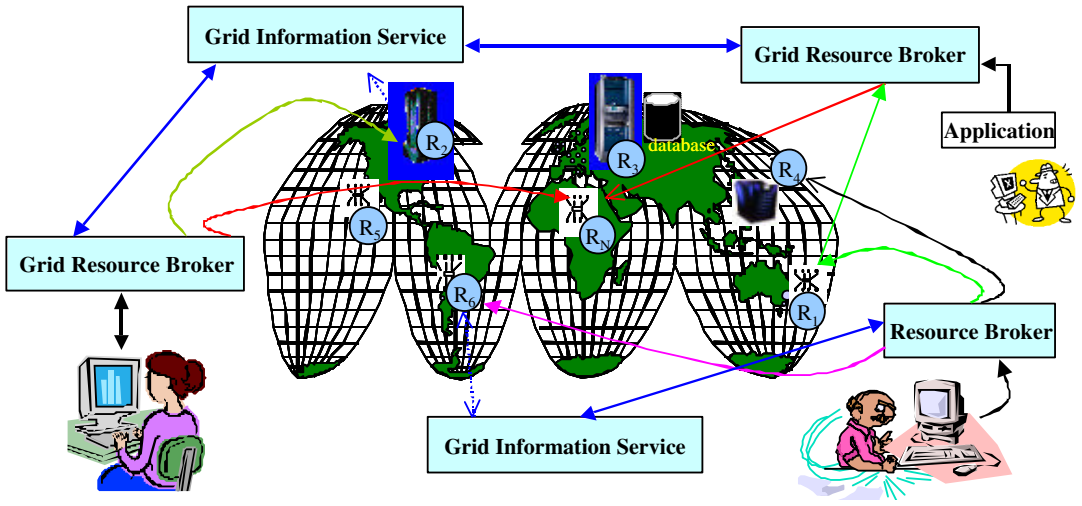


Figure 1. A generic view of the World-Wide Grid computing environment.

Therefore, in [6–8], we investigated the use of economics as a metaphor for management of resources in Grid computing environments.

A Grid resource broker, called Nimrod-G [5], has been developed that performs scheduling of parameter sweep, task-farming applications on geographically distributed resources. It supports deadline and budget-based scheduling driven by market-based economic models. To meet users' quality of service requirements, our broker dynamically leases Grid resources and services at runtime depending on their capability, cost, and availability. Many scheduling experiments have been conducted on the execution of data-intensive, science applications such as molecular modeling for drug design under a few Grid scenarios (like 2 h deadline and 10 machines for a single user). The ability to experiment with a large number of Grid scenarios was limited by the number of resources that were available in the WWG (World-Wide Grid) testbed [9]. Also, it was impossible to create a *repeatable* and *controlled* environment for experimentation and evaluation of scheduling strategies. This is because resources in the Grid span across multiple administrative domains, each with their own policies, users, and priorities.

The researchers and students, investigating resource management and scheduling for large-scale distributed computing, need a simple framework for deterministic modeling and simulation of resources and applications to evaluate scheduling strategies. For most who do not have access to ready-to-use testbed infrastructures, building them is expensive and time consuming. Also, even for those who have access, the testbed size is limited to a few resources and domains; and testing scheduling algorithms for scalability and adaptability, and evaluating scheduler performance for various applications and resource scenarios is harder and impossible to trace. To overcome these limitations, we provide a Java-based Grid simulation toolkit called GridSim. The Grid computing

researchers and educators also recognized the importance and the need for such a toolkit for modeling and simulation environments [10]. It should be noted that this paper has a major orientation towards Grid, however, we believe that our discussion and thoughts apply equally well to P2P systems since resource management and scheduling issues in both systems are quite similar.

The GridSim toolkit supports modeling and simulation of a wide range of heterogeneous resources, such as single or multiprocessors, shared and distributed memory machines such as PCs, workstations, SMPs, and clusters with different capabilities and configurations. It can be used for modeling and simulation of application scheduling on various classes of parallel and distributed computing systems such as clusters [11], Grids [1], and P2P networks [2]. The resources in clusters are located in a single administrative domain and managed by a single entity, whereas in Grid and P2P systems, resources are geographically distributed across multiple administrative domains with their own management policies and goals. Another key difference between cluster and Grid/P2P systems arises from the way application scheduling is performed. The *schedulers* in cluster systems focus on enhancing overall system performance and utility, as they are responsible for the whole system. In contrast, schedulers in Grid/P2P systems called *resource brokers*, focus on enhancing performance of a specific application in such a way that its end-users' requirements are met.

The GridSim toolkit provides facilities for the modeling and simulation of resources and network connectivity with different capabilities, configurations, and domains. It supports primitives for application composition, information services for resource discovery, and interfaces for assigning application tasks to resources and managing their execution. These features can be used to simulate resource brokers or Grid schedulers for evaluating performance of scheduling algorithms or heuristics. We have used the GridSim toolkit to create a resource broker that simulates Nimrod-G for design and evaluation of deadline and budget constrained scheduling algorithms with cost and time optimizations.

The rest of this paper is organized as follows. Section 2 discusses related work with highlights on unique features that distinguish our toolkit from other packages. The GridSim architecture and internal components that make up GridSim simulations are discussed in Section 3. Section 4, discusses how to build GridSim based scheduling simulations. Sample results of simulation of a resource broker similar to Nimrod-G with a deadline and budget constrained cost-optimization scheduling algorithm is discussed in Section 5. The final section summarizes the paper along with suggestions for future works.

2. RELATED WORK

Simulation has been used extensively for modeling and evaluation of real world systems, from business process and factory assembly lines to computer systems design. Accordingly, over the years, modeling and simulation has emerged as an important discipline and many standard and application-specific tools and technologies have been built. They include simulation languages (e.g. Simscript [12]), simulation environments (e.g. Parsec [13]), simulation libraries (SimJava [14]), and application specific simulators (e.g. OMNet++ network simulator [15]). While a large body of knowledge and tools exists, there are very few tools available for application scheduling simulation in Grid computing environments. The notable ones are: Bricks [16], MicroGrid [17], SimGrid [18], and our GridSim toolkit.

The Bricks simulation system [16], developed at the Tokyo Institute of Technology in Japan, helps in simulating client-server like global computing systems that provide remote access to scientific libraries

and packages running on high-performance computers. It follows centralized global scheduling methodology as opposed to our work in which each application scheduling is managed by the users' own resource broker.

The MicroGrid emulator [17], undertaken at the University of California at San Diego (UCSD), is modeled after Globus [19]. It allows execution of applications constructed using the Globus toolkit in a controlled virtual Grid emulated environment. The results produced by emulation can be precise, but modeling numerous applications, Grid environments, and scheduling scenarios for realistic statistical analysis of scheduling algorithms is time consuming as applications run on emulated resources. Also, scheduling algorithms, designers generally work with application models instead of constructing actual applications. Therefore, MicroGrid's need for an application constructed using Globus imposes significant development overhead. However, when an actual system is implemented by incorporating scheduling strategies that are evaluated using simulation, the MicroGrid emulator can be used as a complementary tool for verifying simulation results with real applications.

The SimGrid toolkit [18], developed at UCSD, is a C language based toolkit for the simulation of application scheduling. It supports modeling of resources that are *time-shared* and the load can be injected as constants or from real traces. It is a powerful system that allows creation of tasks in terms of their execution time and resources with respect to a standard machine capability. Using SimGrid APIs, tasks can be assigned to resources depending on the scheduling policy being simulated. It has been used for a number of real studies, and demonstrates the power of simulation. However, because SimGrid is restricted to a single scheduling entity and time-shared systems, it is difficult to simulate multiple competing users, applications, and schedulers, each with their own policies when operating under a market-like Grid computing environment, without extending the toolkit substantially. Also, many large-scale resources in the Grid environment are space-shared machines and they need to be supported in simulation. Hence, our GridSim toolkit extends the ideas in existing systems and overcomes their limitations accordingly.

Finally, we have chosen to implement GridSim in Java by leveraging SimJava's [14] basic discrete event simulation infrastructure. This feature is likely to appeal to educators and students since Java has emerged as a popular programming language for network computing.

3. GridSim: GRID MODELING AND SIMULATION TOOLKIT

The GridSim toolkit provides a comprehensive facility for simulation of different classes of heterogeneous resources, users, applications, resource brokers, and schedulers. It can be used to simulate application schedulers for single or multiple administrative domain distributed computing systems such as clusters and Grids. Application schedulers in the Grid environment, called resource brokers, perform resource discovery, selection, and aggregation of a diverse set of distributed resources for an individual user. This means that each user has his or her own private resource broker and hence it can be targeted to optimize for the requirements and objectives of its owner. In contrast, schedulers, managing resources such as clusters in a single administrative domain, have complete control over the policy used for allocation of resources. This means that all users need to submit their jobs to the *central* scheduler, which can be targeted to perform global optimization such as higher system utilization and overall user satisfaction depending on resource allocation policy or optimize for high priority users.

3.1. Features

Salient features of the GridSim toolkit include the following.

- It allows modeling of heterogeneous types of resources.
- Resources can be modeled operating under space- or time-shared mode.
- Resource capability can be defined (in the form of MIPS (Million Instructions Per Second) as per SPEC (Standard Performance Evaluation Corporation) benchmark).
- Resources can be located in any time zone.
- Weekends and holidays can be mapped depending on resource's local time to model non-Grid (local) workload.
- Resources can be booked for advance reservation.
- Applications with different parallel application models can be simulated.
- Application tasks can be heterogeneous and they can be CPU or I/O intensive.
- There is no limit on the number of application jobs that can be submitted to a resource.
- Multiple user entities can submit tasks for execution simultaneously in the same resource, which may be time-shared or space-shared. This feature helps in building schedulers that can use different market-driven economic models for selecting services competitively.
- Network speed between resources can be specified.
- It supports simulation of both static and dynamic schedulers.
- Statistics of all or selected operations can be recorded and they can be analyzed using GridSim statistics analysis methods.

3.2. System architecture

We employed a layered and modular architecture for Grid simulation to leverage existing technologies and manage them as separate components. A multi-layer architecture and abstraction for the development of GridSim platform and its applications is shown in Figure 2. The first layer is concerned with the scalable Java interface and the runtime machinery, called JVM (Java Virtual Machine), whose implementation is available for single and multiprocessor systems including clusters [20]. The second layer is concerned with a basic discrete-event infrastructure built using the interfaces provided by the first layer. One of the popular discrete-event infrastructure implementations available in Java is SimJava [14]. Recently, a distributed implementation of SimJava was also made available. The third layer is concerned with modeling and simulation of core Grid entities such as resources, information services, and so on; application model, uniform access interface, and primitives application modeling and framework for creating higher level entities. The GridSim toolkit focuses on this layer that simulates system entities using the discrete-event services offered by the lower-level infrastructure. The fourth layer is concerned with the simulation of resource aggregators called Grid resource brokers or schedulers. The final layer is focused on application and resource modeling with different scenarios using the services provided by the two lower-level layers for evaluating scheduling and resource management policies, heuristics, and algorithms. In this section, we briefly discuss the SimJava model for discrete events (a second-layer component) and focus mainly on the GridSim (the third layer) design and implementation. Resource broker simulation and performance evaluation are highlighted in the next two sections.

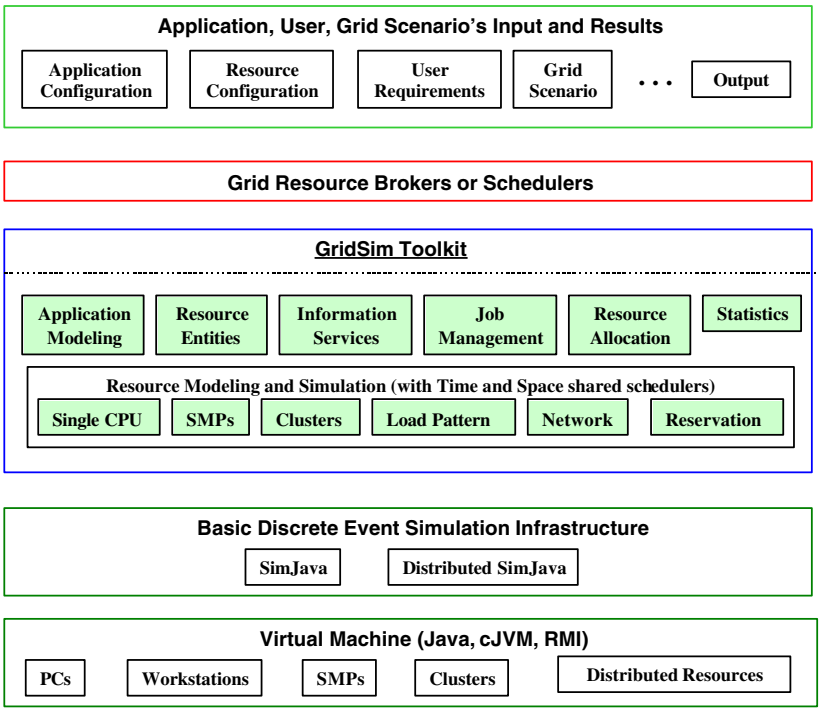


Figure 2. A modular architecture for GridSim platform and components.

3.2.1. SimJava discrete event model

SimJava [14] is a general purpose discrete event simulation package implemented in Java. Simulations in SimJava contain a number of entities, each of which runs in parallel in its own thread. An entity's behaviour is encoded in Java using its body() method. Entities have access to a small number of simulation primitives:

- sim_schedule() sends event objects to other entities via ports;
- sim_hold() holds for some simulation time;
- sim_wait() waits for an event object to arrive.

These features help in constructing a network of active entities that communicate by sending and receiving passive event objects efficiently.

The sequential discrete event simulation algorithm, in SimJava, is as follows. A central object Sim_system maintains a timestamp ordered queue of future events. Initially all entities are created and their body() methods are put in run state. When an entity calls a simulation function, the Sim_system object halts that entity's thread and places an event on the future queue to signify processing

the function. When all entities have halted, Sim_system pops the next event off the queue, advances the simulation time accordingly, and restarts entities as appropriate. This continues until no more events are generated. If the JVM supports native threads, then all entities starting at exactly the same simulation time may run concurrently.

3.2.2. GridSim entities

GridSim supports entities for simulation of single processor and multiprocessor, heterogeneous resources that can be configured as time- or space-shared systems. It allows setting of the clock to different time zones to simulate geographic distribution of resources. It supports entities that simulate networks used for communication among resources. During simulation, GridSim creates a number of multi-threaded entities, each of which runs in parallel in its own thread. An entity's behavior needs to be simulated within its body() method, as dictated by SimJava.

A simulation environment needs to abstract all the entities and their time-dependent interactions in the real system. It needs to support the creation of user-defined time-dependent response functions for the interacting entities. The response function can be a function of the past, current, or both states of entities. GridSim based simulations contain entities for the users, brokers, resources, information service, statistics, and network based I/O, as shown in Figure 3. The design and implementation issues of these GridSim entities are discussed below.

3.2.2.1. User. Each instance of the User entity represents a Grid user. Each user may differ from the rest of users with respect to the following characteristics:

- types of job created, e.g. job execution time, number of parametric replications, etc.;
- scheduling optimization strategy, e.g. minimization of cost, time, or both;
- activity rate, e.g. how often it creates new job;
- time zone; and
- absolute deadline and budget; or
- D- and B-factors, deadline and budget relaxation parameters, measured in the range [0, 1] express deadline and budget affordability of the user relative to the application processing requirements and available resources.

3.2.2.2. Broker. Each user is connected to an instance of the Broker entity. Every job of a user is first submitted to its broker and the broker then schedules the parametric tasks according to the user's scheduling policy. Before scheduling the tasks, the broker dynamically gets a list of available resources from the global directory entity. Every broker tries to optimize the policy of its user and therefore, brokers are expected to face extreme competition while gaining access to resources. The scheduling algorithms used by the brokers must be highly adaptable to the market's supply and demand situation.

3.2.2.3. Resource. Each instance of the Resource entity represents a Grid resource. Each resource may differ from the rest of the resources with respect to the following characteristics:

- number of processors;
- cost of processing;

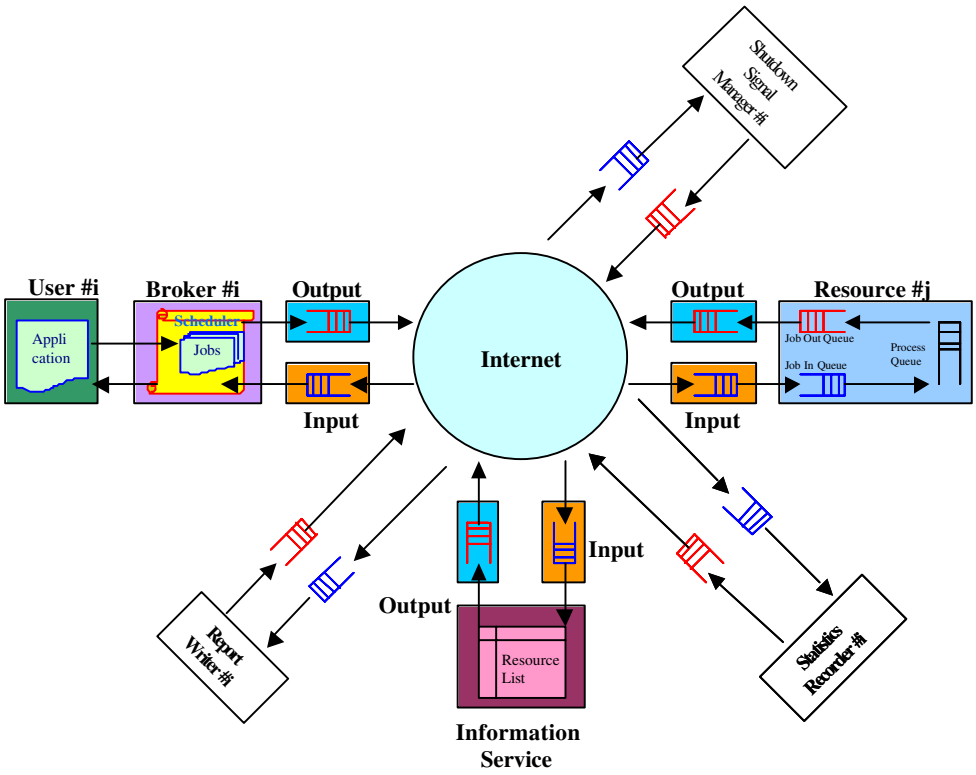


Figure 3. A flow diagram in GridSim based simulations.

- speed of processing;
- internal process scheduling policy, e.g. time-shared or space-shared;
- local load factor; and
- time zone.

The resource speed and the job execution time can be defined in terms of the ratings of standard benchmarks such as MIPS and SPEC. They can also be defined with respect to the standard machine. Upon obtaining the resource contact details from the Grid information service, brokers can query resources directly for their static and dynamic properties.

3.2.2.4. Grid information service. Providing resource registration services and keeping track of a list of resources available in the Grid. The brokers can query this for resource contact, configuration, and status information.

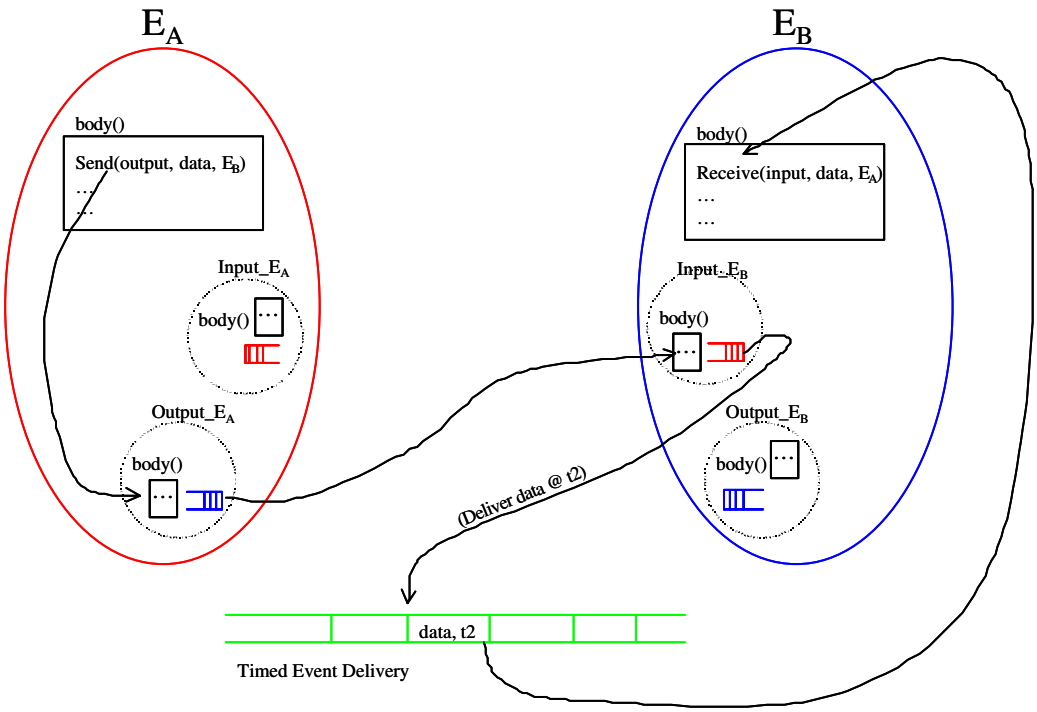


Figure 4. Entity communication model via its Input and Output entities.

3.2.2.5. Input and output. The flow of information among the GridSim entities happens via their Input and Output entities. Every networked GridSim entity has I/O channels or ports, which are used for establishing a link between the entity and its own Input and Output entities. Note that the GridSim entity and its Input and Output entities are threaded entities, i.e. they have their own execution thread with body() method that handles events. The architecture for the entity communication model in GridSim is illustrated in Figure 4. The use of separate entities for input and output enables a networked entity to model full duplex and multi-user parallel communications. The support for buffered input and output channels associated with every GridSim entity provides a simple mechanism for an entity to communicate with other entities and at the same time enables modeling of the necessary communications delay transparently.

3.3. Application model

GridSim does not explicitly define any specific application model. It is up to the developers (of schedulers and resource brokers) to define them. We have experimented with a task-farming application model and we believe that other parallel application models such as process parallelism,

Directed Acyclic Graphs (DAGs), divide and conquer etc., described in [21], can also be modeled and simulated using GridSim.

In GridSim, each independent task may require varying processing time and input files size. Such tasks can be created and their requirements are defined through *Gridlet* objects. A *Gridlet* is a package that contains all the information related to the job and its execution management details such as job length expressed in MIPS, disk I/O operations, the size of input and output files, and the job originator. These basic parameters help in determining execution time, the time required to transport input and output files between users and remote resources, and returning the processed Gridlets back to the originator along with the results. The GridSim toolkit supports a wide range of Gridlet management protocols and services that allow schedulers to map a Gridlet to a resource and manage it throughout the life cycle.

3.4. Interaction protocols model

The protocols for interaction between GridSim entities are implemented using events. In GridSim, entities use events for both service request and service delivery. The events can be raised by any entity to be delivered immediately or with specified delay to other entities or itself. The events that are originated from the same entity are called *internal events* and those originated from the external entities are called *external events*. Entities can distinguish these events based on the source identification associated with them. The GridSim protocols are used for defining entity services. Depending on the service protocols, the GridSim events can be further classified into *synchronous* and *asynchronous* events. An event is called *synchronous* when the event source entity waits until the event destination entity performs all the actions associated with the event (i.e. the delivery of full service). An event is called *asynchronous* when the event source entity raises an event and continues with other activities without waiting for its completion. When the destination entity receives such events or service requests, it responds back with results by sending one or more events, which can then take appropriate actions. It should be noted that external events could be synchronous or asynchronous, but internal events need to be raised as asynchronous events only to avoid deadlocks.

A complete set of entities in a typical GridSim simulation and the use of events for simulating interaction between them are shown in Figures 5 and 6. Figure 5 emphasizes the interaction between a resource entity that simulates time-shared scheduling and other entities. Figure 6 emphasizes the interaction between a resource entity that simulates a space-shared system and other entities. In this section we briefly discuss the use of events for simulating Grid activities.

The GridSim entities (user, broker, resource, information service, statistics, shutdown, and report writer) send events to other entities to signify the request for service, to deliver results, or to raise internal actions. Note that GridSim implements core entities that simulate resource, information service, statistics, and shutdown services. These services are used to simulate a user with application, a broker for scheduling, and an optional report writer for creating statistical reports at the end of a simulation. The event source and destination entities must agree upon the protocols for service request and delivery. The protocols for interaction between the user-defined and core entities are pre-defined.

When GridSim starts, the resource entities register themselves with the Grid Information Service (GIS) entity, by sending events. This resource registration process is similar to GRIS (Grid Resource Information Server) registering with GIIS (Grid Index Information Server) in the Globus system. Depending on the user entity's request, the broker entity sends an event to the GIS entity, to signify a

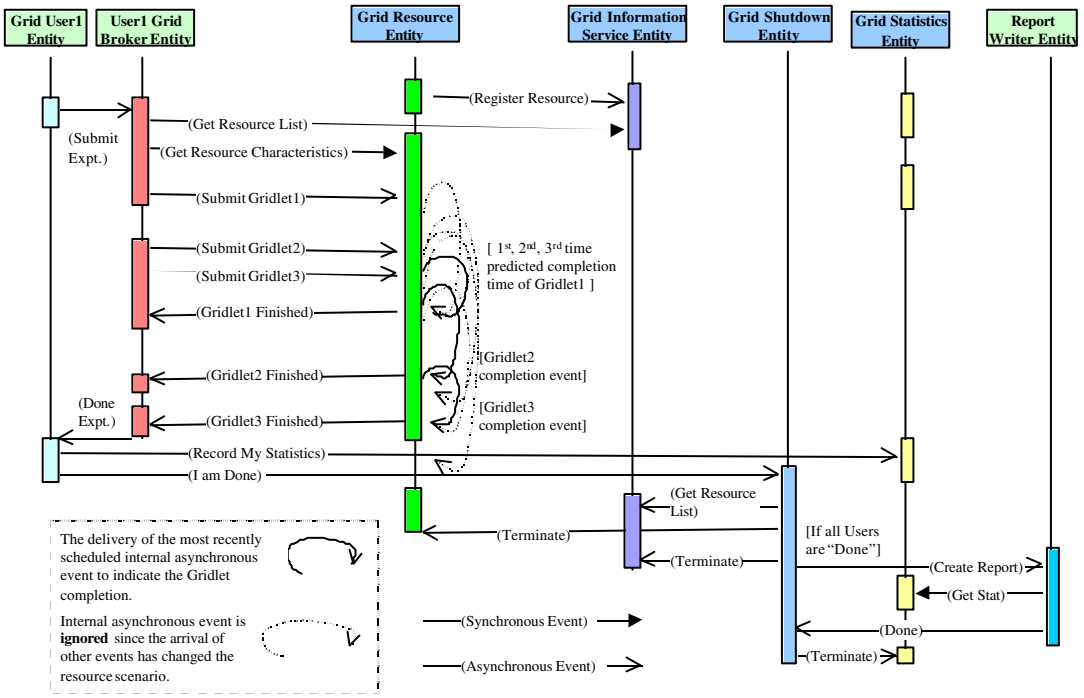


Figure 5. An event diagram for the interaction between a time-shared resource and other entities.

query for resource discovery. The GIS entity returns a list of registered resources and their contact details. The broker entity sends events to resources with a request for resource configuration and properties. They respond with dynamic information such as resources cost, capability, availability, load, and other configuration parameters. These events involving the GIS entity are synchronous in nature.

Depending on the resource selection and scheduling strategy, the broker entity places asynchronous events for resource entities in order to dispatch Gridlets for execution—the broker need not wait for a resource to complete the assigned work. When the Gridlet processing is finished, the resource entity updates the Gridlet status and processing time and sends it back to the broker by raising an event to signify its completion.

The GridSim resources use internal events to simulate resource behavior and resource allocation. The entity needs to be modeled in such a way that it is able to receive all events meant for it. However, it is up to the entity to decide on the associated actions. For example, in time-shared resource simulations (see Figure 5) internal events are scheduled to signify the completion time of a Gridlet, which has the smallest remaining processing time requirement. Meanwhile, if an external event arrives, it changes the share resource availability for each Gridlet, which means the most recently scheduled event may

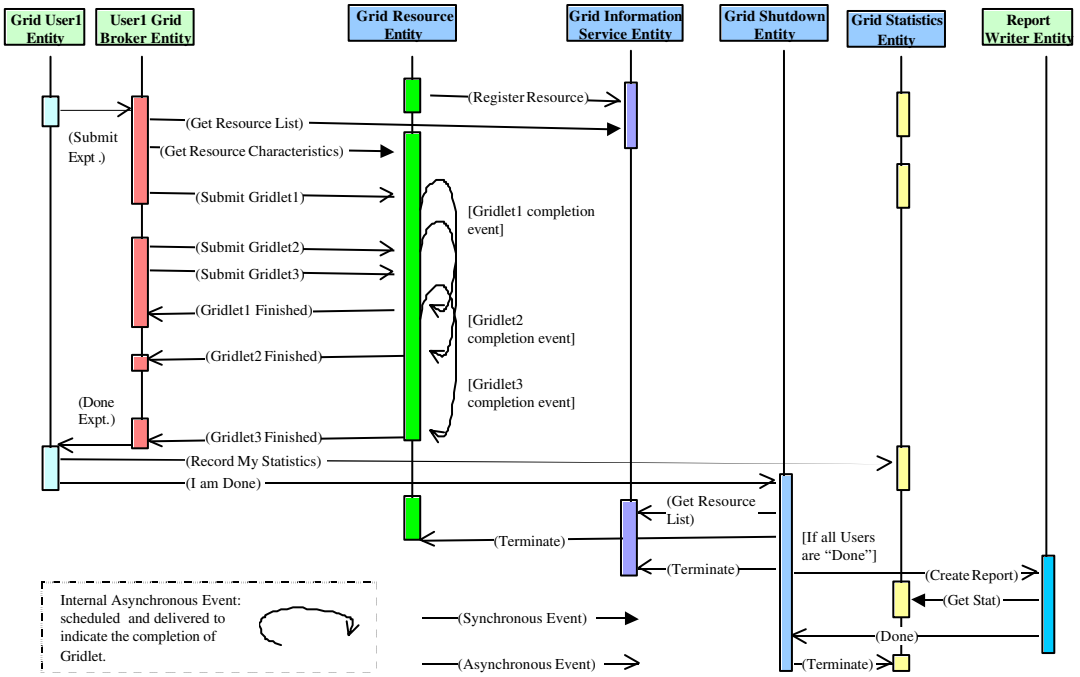


Figure 6. An event diagram for the interaction between a space-shared resource and other entities.

not necessarily signify the completion of a Gridlet. The resource entity can discard such internal events without processing. The use of internal events for simulating resources is discussed in detail in Section 3.5.

3.5. Resource model—simulating multitasking and multiprocessing

In the GridSim toolkit, we can create Processing Elements (PEs) with different speeds (measured in either MIPS or SPEC-like ratings). Then, one or more PEs can be put together to create a machine. Similarly, one or more machines can be put together to create a Grid resource. Thus, the resulting Grid resource can be a single processor, shared memory multiprocessors (SMP), or a distributed memory cluster of computers. These Grid resources can simulate time- or space-shared scheduling depending on the allocation policy. A single PE or SMP-type Grid resource is typically managed by time-shared operating systems that use a round-robin scheduling policy (see Figure 9) for multitasking. The distributed memory multiprocessing systems (such as clusters) are managed by queuing systems, called space-shared schedulers, that execute a Gridlet by running it on a dedicated PE (see Figure 12) when allocated. The space-shared systems use resource allocation policies such as

first-come-first-served (FCFS), back filling, shortest-job-first-served (SJFS), and so on. It should also be noted that resource allocation within high-end SMPs could also be performed using the space-shared schedulers.

Multitasking and multiprocessing systems allow concurrently running tasks to share system resources such as processors, memory, storage, I/O, and network by scheduling their use for very short time intervals. A detailed simulation of scheduling tasks in the real systems would be complex and time consuming. Hence, in GridSim, we abstract these physical entities and simulate their behavior using process oriented, discrete event 'interrupts' with a time interval as large as the time required for the completion of the smallest remaining-time job. The GridSim resources can send, receive, or schedule events to simulate the execution of jobs. It schedules self-events for simulating resource allocation depending on the scheduling policy and the number of jobs in the queue or in execution.

Let us consider the following scenario to illustrate the simulation of Gridlets execution and scheduling within a GridSim resource. A resource consists of two shared or distributed memory PEs each with a MIPS rating of 1, for simplicity. Three Gridlets that represent jobs with processing requirements equivalent to 10, 8.5, and 9.5 MI (million instructions) arrive in simulation times 0, 4, and 7, respectively. The way GridSim schedules jobs to PEs is shown schematically in Figure 9 for time-shared resources and Figure 12 for space-shared resources.

3.5.1. Simulation of scheduling in time-shared resources

The GridSim resource simulator uses internal events to simulate the execution and allocation of PEs' share to Gridlet jobs. When jobs arrive, time-shared systems start their execution immediately and share resources among all jobs. Whenever a new Gridlet job arrives, we update the processing time of existing Gridlets and then add this newly arrived job to the execution set. We schedule an internal event to be delivered at the earliest completion time of the smallest job in the execution set. It then waits for the arrival of events.

A complete algorithm for simulation of time-share scheduling and execution is shown in Figure 7. If a newly arrived event happens to be an internal event whose tag number is the same as the most recently scheduled event, then it is recognized as a job completion event. Depending on the number of Gridlets in execution and the number of PEs in a resource, GridSim allocates the appropriate PE share to all Gridlets for the event duration using the algorithm shown in Figure 8. It should be noted that Gridlets sharing the same PE would get an equal amount of PE share. The completed Gridlet is sent back to its originator (broker or user) and removed from the execution set. GridSim schedules a new internal event to be delivered at the forecasted earliest completion time of the remaining Gridlets.

Figure 9 illustrates the simulation of the time-share scheduling algorithm and the Gridlets' execution. When Gridlet1 arrives at time 0, it is mapped to PE1 and an internal event to be delivered at time 10 is scheduled since the predicted completion time is still 10. At time 4, Gridlet2 arrives and it is mapped to PE2. The completion time of Gridlet2 is predicted as 12.5 and the completion time of Gridlet1 is still 10 since both of them are executing on different PEs. A new internal event is scheduled, which will still be delivered at time 10. At time 7, Gridlet3 arrives, which is mapped to PE2. It shares the PE time with Gridlet2. At time 10, an internal event is delivered to the resource to signify the completion of Gridlet1, which is then sent back to the broker. At this moment, as the number of Gridlets is equal the number of PEs, they are mapped to different PEs. An internal event to be delivered at time 14 is

Algorithm: Time-Shared Grid Resource Event Handler()

1. Wait for an event
2. If the external and Gridlet arrival event, then:
   BEGIN /*a new job arrived*/
   a. Allocate PE Share for Gridlets Processed so far
   b. Add arrived Gridlet to Execution_Set
   c. Forecast completion time of all Gridlets in Execution_Set
   d. Schedule an event to be delivered at the smallest completion time
   END
3. If event is internal and its tag value is the same as the recently scheduled internal event tag,
   BEGIN /*a job finish event*/
   a. Allocate PE Share for Gridlets Processed so far
   b. Update finished Gridlet's PE and Wall clock time parameters and send it back to the broker
   c. Remove finished Gridlet from the Execution_Set and add to Finished_Set
   d. Forecast completion time of all Gridlets in Execution_Set
   e. Schedule an event to be delivered at the smallest completion time
   END
4. Repeat the above steps until the end of simulation event is received

Figure 7. An event handler for simulating time-shared resource scheduling.

Algorithm: PE_Share_Allocation(Duration)
BEGIN
1. Identify total MI per PE for the duration and the number of PEs that process one extra Gridlet
   TotalMIperPE = MIPSRatingOfOnePE()*Duration
   MinNoOfGridletsPerPE = NoOfGridletsInExec / NoOfPEs
   NoofPEsRunningOneExtraGridlet = NoOfGridletsInExec % NoOfPEs
2. Identify maximum and minimum MI share that Gridlet get in the Duration
   If(NoOfGridletsInExec <= NoOfPEs), then:
      MaxSharePerGridlet = MinSharePerGridlet = TotalMIperPE
      MaxShareNoOfGridlets = NoOfGridletsInExec
   Else /* NoOfGridletsInExec > NoOfPEs */
      MaxSharePerGridlet = TotalMIperPE/MinNoOfGridletsPerPE
      MinSharePerGridlet = TotalMIperPE/(MinNoOfGridletsPerPE+1)
      MaxShareNoOfGridlets = (NoOfPEs - NoOfPEsRunningOneExtraGridlet)* MinNoOfGridletsPerPE
END

Figure 8. PE share allocation to Gridlet in time-shared GridSim resource.

scheduled to indicate the predicted completion time of Gridlet2. As simulation proceeds, an internal event is delivered at time 14 and Gridlet2 is sent back to the broker. An internal event to be delivered at time 18 is scheduled to indicate the predicted completion time of Gridlet3. Since there were no other Gridlets submitted before this time, the resource receives an internal interrupt at time 18, which signifies the completion of Gridlet3. A schematic representation of the Gridlets' arrival, internal events delivery, and sending them back to the broker is shown in Figure 5. Detailed statistical data on the arrival, execution start, finish, and elapsed time of all Gridlets are shown in Table I.

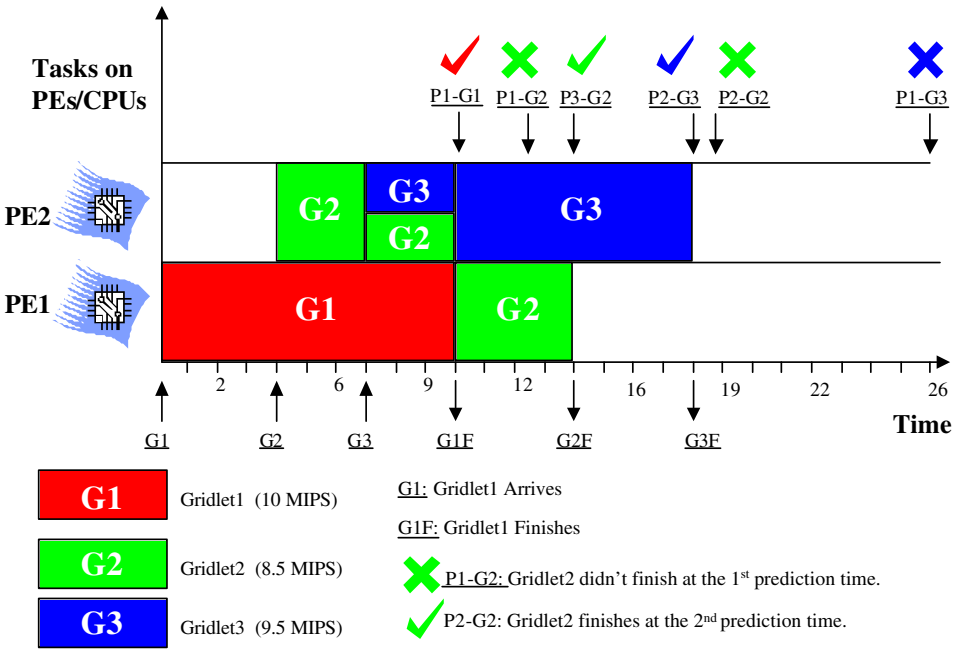


Figure 9. Modeling time-shared multitasking and multiprocessing based on an event scheme.

Table I. A scheduling statistics scenario for time- and space-shared resources in GridSim.

			Time-shared resource			Space-shared resource		
Gridlet number	Length (MI)	Arrival time (a)	Start time (s)	Finish time (f)	Elapsed time ($f - a$)	Start time (s)	Finish time (f)	Elapsed time ($f - a$)
G1	10	0	0	10	10	0	10	10
G2	8.5	4	4	14	10	4	12.5	8.5
G3	9.5	7	7	18	11	10	19.5	12.5

Algorithm: Space-Shared Grid Resource Event Handler()

1. Wait for an event and Identify Type of Event received
2. If it external and Gridlet arrival event, then:
   BEGIN /* a new job arrived */
   - If the number of Gridlets in execution are less than the number of PEs in the resource, then
     Allocate_PE_to_the_Gridlet() /* It should schedule an Gridlet completion event */
   - If not, Add Gridlet to the Gridlet_Submitted_Queue

   END
3. If event is internal and its tag value is the same recently scheduled internal event tag,
   BEGIN /* a job finish event */
   - Update finished Gridlet's PE and Wall clock time parameters and send it back to the broker.
   - Set the status of PE to FREE.
   - Remove finished Gridlet from the Execution_Set and add to the Finished_Set.
   - If Gridlet_Submitted_Queue has Gridlets in waiting, then
     Choose the Gridlet to be Processed() /* e.g., first one in Q if FCFS policy is used */
     Allocate_PE_to_the_Gridlet() /* It should schedule a Gridlet completion event */

   END
4. Repeat above steps until end of simulation event is received

Figure 10. An event handler for simulating space-shared resource scheduling.

3.5.2. Simulation of scheduling in space-shared resources

The GridSim resource simulator uses internal events to simulate the execution and allocation of PEs to Gridlet jobs. When a job arrives, space-shared systems start its execution immediately if there is a free PE available, otherwise, it is queued. During the Gridlet assignment, job-processing time is determined and the event is scheduled for delivery at the end of the execution time. Whenever a Gridlet job finishes, an internal event is delivered to signify the completion of the scheduled Gridlet job. The resource simulator then frees the PE allocated to it and checks if there are any other jobs waiting in the queue. If there are jobs waiting in the queue, then it selects a suitable job depending on the policy and assigns it to the PE which is free.

A complete algorithm for simulation of space-share scheduling and execution is shown in Figure 10. If a newly arrived event happens to be an internal event whose tag number is the same as the most recently scheduled event, then it is recognized as a Gridlet completion event. If there are Gridlets in the submission queue, then depending on the allocation policy (e.g. the first Gridlet in the queue if FCFS policy is used), GridSim selects a suitable Gridlet from the queue and assigns it to the PE or a suitable PE if more than one PE is free. See Figure 11 for an illustration of the allocation of PEs to Gridlets. The completed Gridlet is sent back to its originator (broker or user) and removed from the execution set. GridSim schedules a new internal event to be delivered at the completion time of the scheduled Gridlet job.

Figure 12 illustrates simulation of the space-shared scheduling algorithm and Gridlet execution. When Gridlet1 arrives at time 0, it is mapped to PE1 and an internal event to be delivered at time 10 is scheduled since the predicted completion time is still 10. At time 4, Gridlet2 arrives and it is mapped to

Algorithm: Allocate_PE_to_the_Gridlet(Gridlet gl)
BEGIN
1. Identify a suitable Machine with Free PE
2. Identify a suitable PE in the machine and Assign to the Gridlet
3. Set Status of the Allocated PE to BUSY
4. Determine the Completion Time of Gridlet and Set and internal event to be delivered at the completion time

END

Figure 11. PE allocation to the Gridlets in the space-shared GridSim resource.

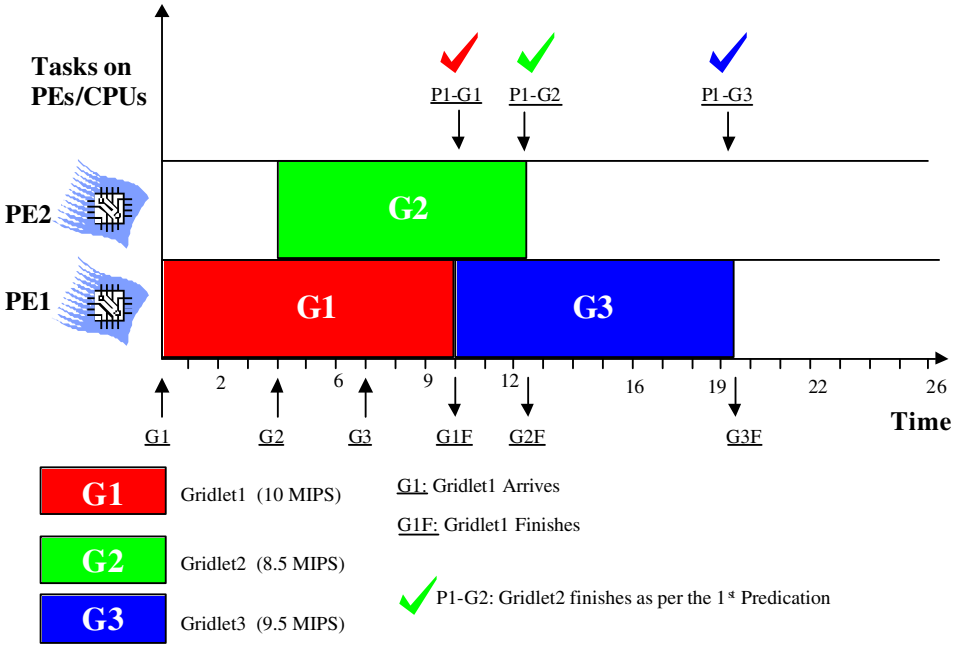


Figure 12. Modeling space-shared multiprocessing based on an event scheme.

PE2. The completion time of Gridlet2 is predicted as 12.5 and the completion time of Gridlet1 is still 10 since both of them are executing on different PEs. A new internal event to be delivered at time 12.5 is scheduled to signify the completion of Gridlet2. At time 7, Gridlet3 arrives. Since there is no free PE available on the resource, it is put into the queue. The simulation continues, i.e. the GridSim resource waits for the arrival of a new event. At time 10 a new event is delivered which happens to signify the completion of Gridlet1, which is then sent back to the broker. It then checks to see if there are any Gridlets waiting in the queue and chooses a suitable Gridlet (in this case Gridlet2, based on FCFS policy) and assigns the available PE to it. An internal event to be delivered at time 19.5 is scheduled to indicate the completion time of Gridlet3 and then waits for the arrival of new events. A new event is delivered at the simulation time 12.5, which signifies the completion of Gridlet2, which is then sent back to the broker. There is no Gridlet waiting in the queue, so it proceeds without scheduling any events and waits for the arrival of the next event. A new internal event arrives at the simulation time 19.5, which signifies the completion of Gridlet3. This process continues until resources receive an external event indicating the termination of simulation. A schematic representation of the arrival of the Gridlets, internal events delivery, and sending them back to the broker is shown in Figure 6. A detailed statistical data on the arrival, execution start, finish, and elapsed time of all Gridlets are shown in Table I.

For every Grid resource, the non-Grid (local) workload is estimated based on typically observed load conditions depending on the time zone of the resource. The network communication speed between a user and the resources is defined in terms of a data transfer speed (baud rate).

3.6. GridSim Java package design

A class diagram hierarchy of the GridSim package, represented using unified modeling language (UML) notation, is shown in Figure 13. The specification of each class contains up to three parts: attributes, methods, and internal classes. In the class diagram, attributes and methods are prefixed with characters '+', '-', and '#', indicating access modifiers public, private, and protected, respectively. The GridSim package implements the following classes.

`class gridsim.Input`: this class extends the eduni.simjava.Sim_entity class. It defines a port through which a simulation entity receives data from the simulated network. It maintains an event queue to serialize the data-in-flow and delivers to its parent entity. Simultaneous inputs can be modeled using multiple instances of this class.

`class gridsim.Output`: this class is very similar to the gridsim.Input class and it defines a port through which a simulation entity sends data to the simulated network. It maintains an event queue to serialize the data-out-flow and delivers to the destination entity. Simultaneous outputs can be modeled by using multiple instances of this class.

`class gridsim.GridSim`: this is the main class of the GridSim package that must be extended by GridSim entities. It inherits event management and threaded entity features from the eduni.simjava.Sim_entity class. The GridSim class adds networking and event delivery features, which allow synchronous or asynchronous communication for service access or delivery. All classes that extend the GridSim class must implement a method called 'body()', which is automatically invoked since it is expected to be responsible for simulating entity behavior.

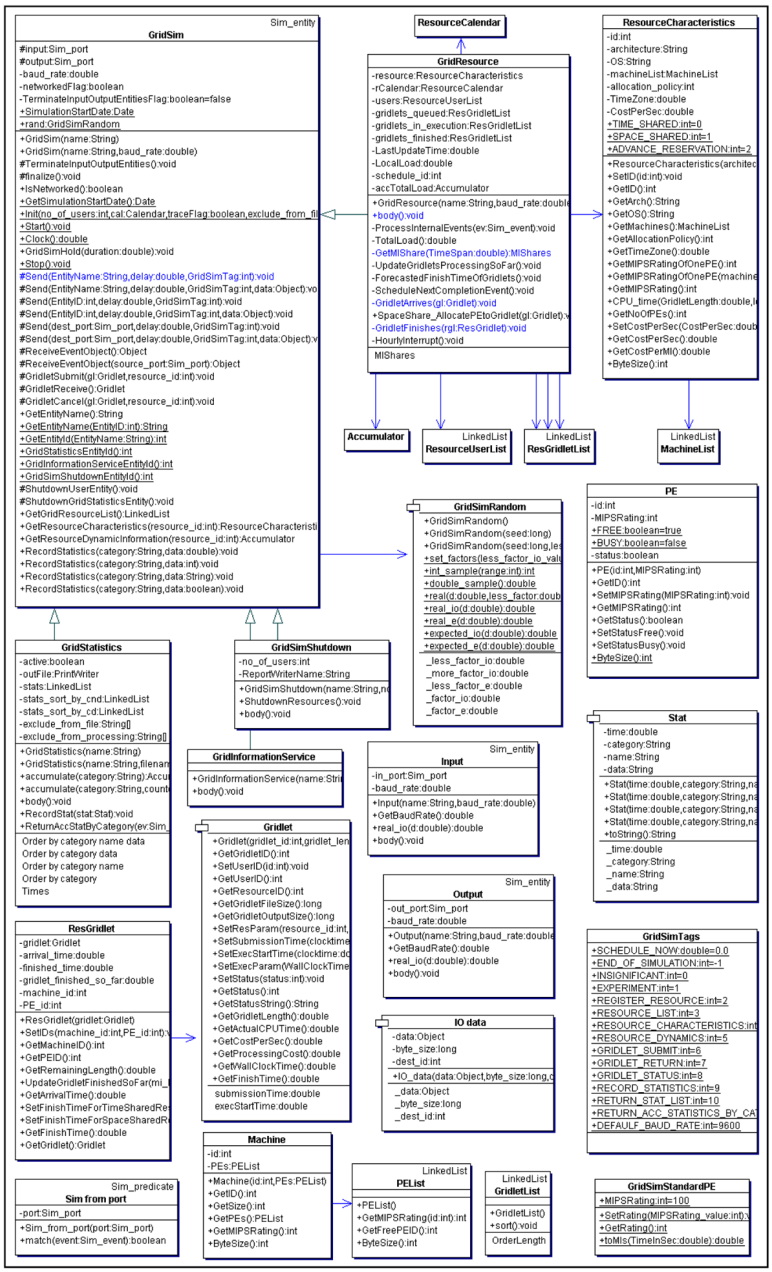


Figure 13. A class hierarchy diagram of the GridSim package.

The entities that extend the GridSim class can be instantiated with or without networked I/O ports. A networked GridSim entity gains communication capability via the objects of GridSim's I/O entity classes, gridsim.Input and gridsim.Output classes. Each I/O entity will have a unique name assuming each GridSim entity that the user creates has a unique name. For example, a resource entity with the name 'Resource2' will have an input entity whose name is prefixed with 'Input_', making the input entity's full name 'Input_Resource2', which is expected to be unique. The I/O entities are concurrent entities, but they are visible within the GridSim entity and are able to communicate with other GridSim entities by sending messages.

The GridSim class supports methods for simulation initialization, management, and flow control. The GridSim environment must be initialized to set-up the simulation environment before creating any other GridSim entities at the user level. This method also prepares the system for simulation by creating three GridSim internal entities—GridInformationService, GridSimShutdown, and GridStatistics. As explained in Section 3.2, the GridInformationService entity simulates the directory that dynamically keeps a list of resources available in the Grid. The GridSimShutdown entity helps in wrapping up a simulation by systematically closing all the opened GridSim entities. The GridStatistics entity provides standard services during the simulation to accumulate statistical data. Invoking the GridSim.Start () method starts the Grid simulation. All the resource and user entities must be instantiated in between invoking the above two methods.

The GridSim class supports static methods for sending and receiving messages between entities directly or via network entities, managing and accessing handles to various GridSim core entities, and recording statistics.

`class gridsim.PE`: this is used to represent CPU/PE, the capability of which is defined in terms of MIPS rating.

`class gridsim.PEList`: maintains a list of PEs that make up a machine.

`class gridsim.Machine`: represents a uniprocessor or shared memory multiprocessor machine.

`class gridsim.MachineList`: an instance of this class simulates a collection of machines. It is up to the GridSim users to define the connectivity among the machines in a collection. Therefore, this class can be instantiated to model simple LAN to cluster to WAN.

`class gridsim.ResourceCharacteristics`: this represents static properties of a resource such as resource architecture, OS, management policy (time- or space-shared), cost, and time zone at which the resource is located along resource configuration.

`class gridsim.GridResource`: extends the GridSim class and gains communication and concurrent entity capability. An instance of this class simulates a resource with properties defined in an object of the `gridsim.ResourceCharacteristics` class. The process of creating a Grid resource is as follows: first create PE objects with a suitable MIPS/SPEC rating, second assemble them together to create a machine. Finally, group one or more objects of the machine to form a resource. A resource having a single machine with one or more PEs is managed as a time-shared system using a round-robin scheduling algorithm. A resource with multiple machines is

treated as a distributed memory cluster and is managed as a space-shared system using FCFS scheduling policy or its variants.

`class gridsim.GridSimStandardPE`: defines MIPS rating for a standard PE or enables the users to define their own MIPS/SPEC rating for a standard PE. This value can be used for creating PEs with relative MIPS/SPEC rating for GridSim resources and creating Gridlets with relative processing requirements.

`class gridsim.ResourceCalendar`: this class implements a mechanism to support modeling a local load on Grid resources that may vary according to the time zone, time, weekends, and holidays.

`class gridsim.GridInformationService`: a GridSim entity that provides Grid resource registration, indexing and discovery services. The Grid resources register their readiness to process Gridlets by registering themselves with this entity. GridSim entities such as the resource broker can contact this entity for resource discovery service, which returns a list of registered resource entities and their contact address. For example, scheduling entities use this service for resource discovery.

`class gridsim.Gridlet`: this class acts as job package that contains job length in MI, the length of input and output data in bytes, execution start and end time, and the originator of the job. Individual users model their application by creating Gridlets for processing them on Grid resources assigned by scheduling entities (resource brokers).

`class gridsim.GridletList`: can be used to maintain a list of Gridlets and support methods for organizing them.

`class gridsim.GridSimTags`: contains various static command tags that indicate a type of action that needs to be undertaken by GridSim entities when they receive events. The different types of tags supported in GridSim along with comments indicating possible purpose are shown in Figure 14.

`class gridsim.ResGridlet`: represents a Gridlet submitted to the resource for processing. It contains a Gridlet object along with its arrival time and the ID of the machine and the PE allocated to it. It acts as a placeholder for maintaining the amount of resource share allocated at various times for simulating time-shared scheduling using internal events.

`class gridsim.GridStatistics`: this is a GridSim entity that records statistical data reported by other entities. It stores data objects with their label and timestamp. At the end of simulation, the user-defined report-writer entity can query recorded statistics of interest for report generation.

`class gridsim.Accumulator`: the objects of this class provide a placeholder for maintaining statistical values of a series of data added to it. It can be queried for mean, sum, standard deviation, and the largest and smallest values in the data series.

`class gridsim.GridSimShutdown`: this is a GridSim entity that waits for termination of all user entities to determine the end of simulation. It then signals the user-defined report-writer


```
public class GridSimTags  {
  public static final double SCHEDULE_NOW = 0.0; // 0.0 indicates NO delay
  public static final int END_OF_SIMULATION = -1;
  public static final int INSIGNIFICANT = 0; // ignore tag
  public static final int EXPERIMENT = 1; // User <-> Broker
  public static final int REGISTER_RESOURCE = 2; // GIS -> ResourceEntity
  public static final int RESOURCE_LIST = 3; // GIS <-> Broker
  public static final int RESOURCE_CHARACTERISTICS = 4; // Broker <-> ResourceEntity
  public static final int RESOURCE_DYNAMICS = 5; // Broker <-> ResourceEntity
  public static final int GRIDLET_SUBMIT = 6; // Broker -> ResourceEntity
  public static final int GRIDLET_RETURN = 7; // Broker <- ResourceEntity
  public static final int GRIDLET_STATUS = 8; // Broker <-> ResourceEntity
  public static final int RECORD_STATISTICS = 9; // Entity -> GridStatistics
  public static final int RETURN_STAT_LIST = 10; // Entity <- GridStatistics
  public static final int RETURN_ACC_STATISTICS_BY_CATEGORY = 11;
  public static final int DEFAULF_BAUD_RATE = 9600; // Default Baud Rate for entities
}
```

Figure 14. Global tags in the GridSim package.

entity to interact with the GridStatistics entity to generate a report. Finally, it signals the end of simulation to other GridSim core entities.

`class gridsim.GridSimRandom`: this class provides static methods for incorporating randomness in data used for any simulation. Any predicted/estimated data, e.g. number of Gridlets used by an experiment, execution time and output size of a Gridlet etc., need to be mapped to real-world data by introducing randomness to reflect the uncertainty that is present in the prediction/estimation process and the randomness that exists in the nature itself. The execution time of a Gridlet on a particular resource, for example, can vary depending on the local load, which is not covered by the scope of the GridSim to simulate.

The real (d, f_L, f_M) method of this class maps the predicted/estimated value d to a random real-world value between $(1-f_L)\times d$ and $(1+f_M)\times d$, using the formula $d\times(1-f_L+(f_L+f_M)\times rd)$ where $0.0 \leqslant f_L, f_M \leqslant 1.0$ and rd is a uniformly distributed `double` value between 0.0 and 1.0. This class also maintains different values of f_L and f_M factors for different situations to represent the different levels of uncertainty involved.

4. BUILDING SIMULATIONS WITH GridSim

To simulate Grid resource brokers using the GridSim toolkit, the developers need to create new entities that exhibit the behavior of Grid users and scheduling systems. The user-defined entities extend the GridSim base class to inherit the properties of concurrent entities capable of communicating with other entities using events. The detailed steps involved in modeling resources and applications, and simulating brokers using the GridSim toolkit are discussed below. We then present the simulation

of a Nimrod-G like resource broker that implements deadline and budget constrained scheduling algorithms.

4.1. A recipe for simulating application scheduling

In this section we present high-level steps, with sample code clips, to demonstrate how GridSim can be used to simulate a Grid environment to analyze scheduling algorithms.

- First, we need to create Grid resources of different capabilities and configurations (a single or multiprocessor with time/space-shared resource manager) similar to those present in the WWG testbed [8]). We also need to create users with different requirements (application and quality of service requirements). A sample code for creating a Grid environment is given in Figure 15.
- Second, we need to model applications by creating a number of Gridlets (that appear similar to Nimrod-G jobs) and define all parameters associated with jobs as shown in Figure 16. The Gridlets need to be grouped together depending on the application model.
- Then, we need to create a GridSim user entity that creates and interacts with the resource broker scheduling entity to coordinate execution experiment. It can also directly interact with GIS and resource entities for Grid information and submitting or receiving processed Gridlets. However, for modularity sake, we encourage the implementation of a separate resource broker entity by extending the GridSim class.
- Finally, we need to implement a resource broker entity that performs application scheduling on Grid resources. A sample code for implementing the broker is shown in Figure 17. First, it accesses the GIS, and then inquires for resource capability including cost. Depending on the processing requirements, it develops a schedule for assigning Gridlets to resources and coordinates the execution. The scheduling policies can be systems-centric like those implemented in many Grid systems such as Condor, or user-centric like the Nimrod-G broker's quality of service driven application scheduling algorithms [7].

4.2. Economic Grid resource broker simulation

We used the GridSim toolkit to simulate the Grid environment and a Nimrod-G like deadline and budget constrained scheduling system called the economic Grid resource broker. The simulated Grid environment contains multiple resources and user entities with different requirements. The users create an experiment that contains application specification (a set of Gridlets that represent application jobs with different processing) and quality of service requirements (deadline and budget constraints with optimization strategy). We created two entities that simulate the users and the brokers by extending the GridSim class. When simulated, each user entity having its own application and quality of service requirements creates its own instance of the broker entity for scheduling Gridlets on resources.

4.2.1. Broker architecture

The broker entity architecture along with its interaction flow diagram with other entities is shown in Figure 18. The key components of the broker are the experiment interface, resource discovery and trading, the scheduling flow manager backed with scheduling heuristics and algorithms, the Gridlets


```
public static void CreateSampleGridEnvironement(int no_of_users, int no_of_resources,
  double B_factor, double D_factor, int policy, double how_long, double seed) {
  Calendar now = Calendar.getInstance();

  String ReportWriterName = "MyReportWriter";
  GridSim.Init(no_of_users, calender, true, eff, efp, ReportWriterName);

  String[] category = {"*.USER.TimeUtilization", "*.USER.GridletCompletionFactor",
"*.USER.BudgetUtilization"};

// Create Report Writer Entity and category indicates types of information to be recorded.
  new ReportWriter(ReportWriterName, no_of_users, no_of_resources, ReportFile, category,
report_on_next_row_flag);

  // Create Resources
  for(int i=0; i<no_of_resources; i++) {
     // Create PEs
     PEList peList = new PEList();
     for(int j=0; j<(i*1+1); j++)
        peList.add(new PE(0, 100));

     // Create machine list
     MachineList mList = new MachineList();
     mList.add(new Machine(0, peList));

     // Create a resource containing machines
     ResourceCharacteristics resource = new ResourceCharacteristics("INTEL", "Linux",
         mList, ResourceCharacteristics.TIME_SHARED, 0.0, i*0.5+1.0);
     LinkedList Weekends = new LinkedList();
     Weekends.add(new Integer(Calendar.SATURDAY));
     Weekends.add(new Integer(Calendar.SUNDAY));
     LinkedList Holidays = new LinkedList(); // no holiday is set!

     // Setup resource as simulated entity with a name (e.g. "Resource_1").
     new GridResource("Resource_"+i, 28000.0, seed, resource,
                                          0.0, 0.0, 0.0, Weekends, Holidays);
  }
  Random r = new Random(seed);
  // Create Application, Experiment, and Users
  for(int i=0; i<no_of_users; i++)
  {
    Random r = new Random(seed*997*(1+i)+1);
    GridletList glList = Application1(r); // it creates Gridlets and returns their list
    Experiment expt = new Experiment(0, glList, policy, true, B_factor, D_factor);
    new UserEntity("U"+i, expt, 28000.0, how_long, seed*997*(1+i)+1, i, user_entity_report);
  }
  // Perform Simulation
  GridSim.Start();
}
```

Figure 15. A sample code segment for creating Grid resource and user entities in GridSim.

```
Gridlet gl = new Gridlet(Gridlet_id, Gridlet_length, GridletFileSize,
                         GridletOutputSize);
```

Figure 16. The Gridlet method in GridSim.

dispatcher, and the Gridlets receptor. The following high-level steps describe the functionality of the broker components and their interaction.

1. The user entity creates an experiment that contains an application description (a list of Gridlets to be processed) and user requirements to the broker via the experiment interface.
2. The broker resource discovery and trading module interacts with the GridSim GIS entity to identify contact information of resources and then interacts with resources to establish their configuration and access cost. It creates a broker resource list that acts as a placeholder for maintaining resource properties, a list of Gridlets committed for execution on the resource, and the resource performance data as predicted through the measure and extrapolation methodology.
3. The scheduling flow manager selects an appropriate scheduling algorithm for mapping Gridlets to resources depending on the user requirements. Gridlets that are mapped to specific resource are added to the Gridlets list in the broker resource.
4. For each of the resources, the dispatcher selects the number of Gridlets that can be staged for execution according to the usage policy to avoid overloading resources with single user jobs.
5. The dispatcher then submits Gridlets to resources using the GridSim asynchronous service.
6. When the Gridlet processing completes, the resource returns it to the broker's Gridlet receptor module, which then measures and updates the runtime parameter, *resource or MI share available to the user*. It aids in predicting the job consumption rate for making scheduling decisions.
7. Steps 3–6 continue until all the Gridlets are processed or the broker exceeds deadline or budget limits. The broker then returns updated experimental data along with processed Gridlets back to the user entity.

A class diagram hierarchy of the Grid broker package built using the GridSim toolkit is shown in Figure 19. The Grid broker package implements the following key classes.

`class Experiment`: acts as a placeholder for representing simulation experiment configuration that includes synthesized application (a set of Gridlets stored in GridletList) and user requirements such as D and B factors or deadline and budget constraints, and optimization strategy. It provides methods for updating and querying the experiment parameters and status. The user entity invokes the broker entity and passes its requirements via the experiment object. On receiving an experiment from its user, the broker schedules Gridlets according to the optimization policy set for the experiment.

`class UserEntity`: a GridSim entity that simulates the user. It invokes the broker and passes the user requirements. When it receives the results of application processing, it records parameters of interest with the gridsim.Statistics entity. When it has no more processing requirements, it sends the `END_OF_SIMULATION` event to the gridsim.GridSimShutdown entity.


```
class Broker extends GridSim {
  private Experiment experiment;
  private LinkedList ResIDList;
  private LinkedList BrokerResourceList;

  public Broker(String name, double baud_rate)
  {
    super(name, baud_rate);
    GridletDispatched = 0;
    GridletReturned = 0;
    Expenses = 0.0;
    MaxGridletPerPE = 2;
  }

  ... // Gridlet scheduling flow code at the Grid Resource Broker level

  public void body() {

    Sim_event ev = new Sim_event();
    // Accept User Commands and Process
    for( sim_get_next(ev); ev.get_tag()!=GridSimTags.END_OF_SIMULATION; sim_get_next(ev))
    {
      experiment = (Experiment) ev.get_data();
      int UserEntityID = ev.get_src();

      // Record Experiment Start Time.
      experiment.SetStartTime();

     // Set Gridlets' OwnerID as this BrokerID so that Resources knows where to return them.
      for(int i=0; i<experiment.GetGridletList().size(); i++)
        ((Gridlet) experiment.GetGridletList().get(i)).SetUserID(get_id());

      // RESOURCE DISCOVERY
      ResIDList = (LinkedList) GetGridResourceList();

      // RESOURCE TRADING and SORTING
      // SCHEDULING
      while (glFinishedList.size() < experiment.GetGridletList().size())
      {
        if((GridSim.Clock()>=experiment.GetDeadline())||(Expenses>=experiment.GetBudget()) )
          break;

        scheduled_count = ScheduleAdviser();
        dispatched_count = Dispatcher();
        received_count = Receiver();

        // Heurisitics for deciding hold condition
        if(dispatched<=0 && received<=0 && glUnfinishedList.size()>0)
        {
          double deadline_left = experiment.GetDeadline()-GridSim.Clock();
          GridSimHold(Math.max(deadline_left*0.01, 1.0));
        }
      }
    }
  ... // Code for actual scheduling policy
  ... // Code for dispatch policy
 }
}
```

Figure 17. A sample code segment for creating a Grid resource broker in GridSim.

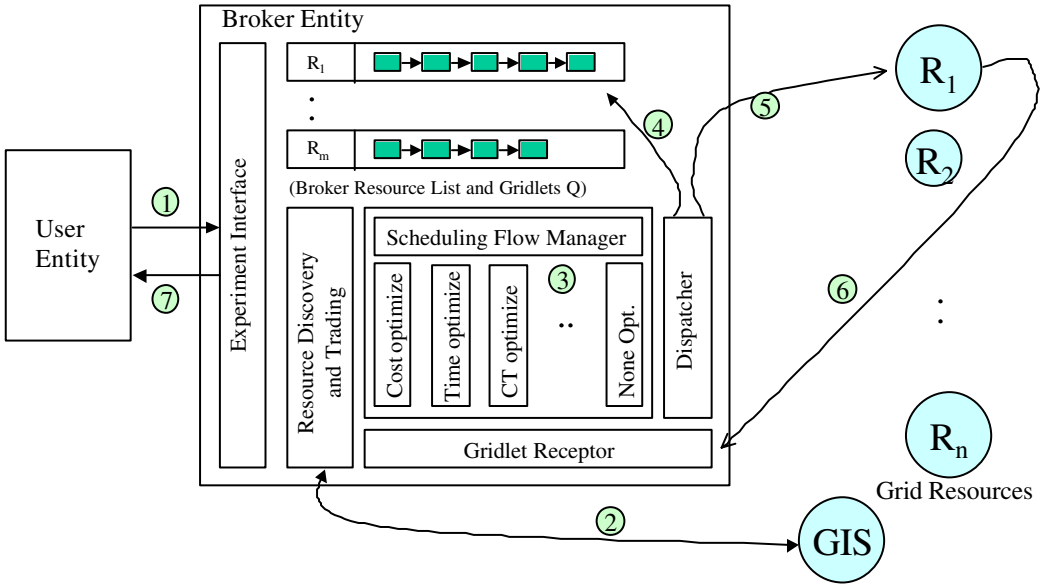


Figure 18. Economic-based Grid resource broker architecture and its interaction with other entities.

`class Broker`: a GridSim entity that simulates the Grid resource broker. On receiving an experiment from the user entity, it carries out resource discovery, and determines deadline and budget values based on D and B factors, and then proceeds with scheduling. It schedules Gridlets on resources depending on user constraints, optimization strategy, and cost of resources and their availability. When it receives the results of application processing, it records parameters of interest with the gridsim.Statistics entity. When it has no more processing requirements, it sends the `END_OF_SIMULATION` event to the gridsim.GridSimShutdown entity. The interaction between the broker and other GridSim entities is shown in Figure 5 for time-shared resources and Figure 6 for space-shared resources.

`class BrokerResource`: acts as a placeholder for the broker to maintain a detailed record on the resources it uses for processing user applications. It maintains resource characteristics, a list of Gridlets assigned to the resource, the actual amount of MIPS available to the user, and a report on the Gridlets processed. These measurements help in extrapolating and predicating the resource performance from the user point of view and aid in scheduling jobs dynamically at runtime.

`class ReportWriter`: a user-defined, optional GridSim entity which is meant for creating a report at the end of each simulation by interacting with the gridsim.Statistics entity. If the user

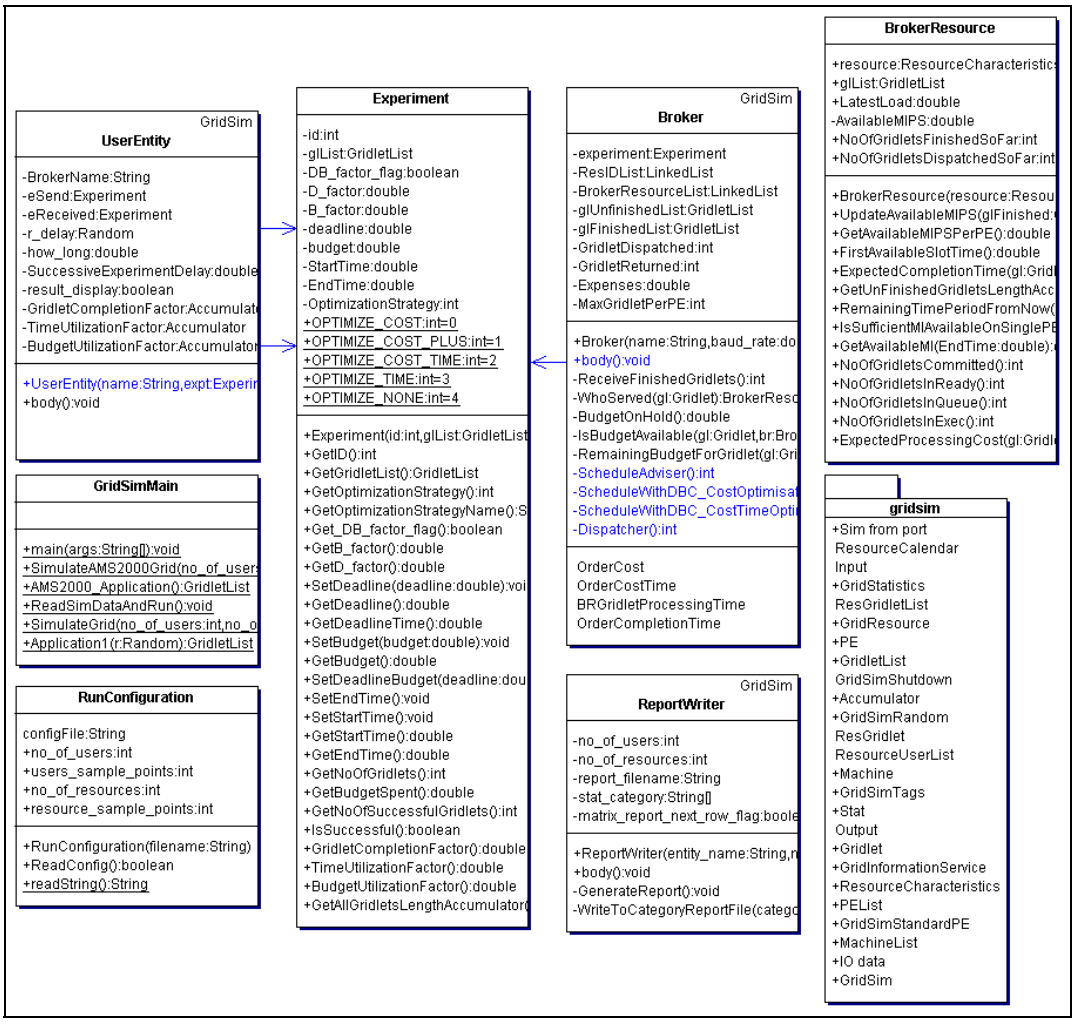


Figure 19. A class hierarchy diagram of the Grid broker using the GridSim package.

Algorithm: DBC_Scheduling_with_Cost_Optimization()

1. RESOURCE DISCOVERY: Identify resources that can be used in this execution with their capability through the Grid Information Service.
2. RESOURCE TRADING: Identify cost of each of the resources in terms of CPU cost per second and capability to be delivered per cost-unit.
3. If the user supplies D and B factors, then determine the absolute deadline and budget based on the capability and cost of resources and user's requirements.
4. SORT resources by increasing order of cost.
5. SCHEDULING: Repeat while there exist unprocessed jobs in application job list with a delay of scheduling event period or occurrence of an event AND the time and process expenses are within deadline and budget limits:
   - [SCHEDULE ADVISOR with Policy]
   - a. For each resource predict and establish the job consumption rate or the available resource share through measure and extrapolation.
   - b. For each resource based on its job consumption rate or available resource share, predict and establish the number of jobs a resource can process by the deadline.
   - c. For each resource in order:
     - i. If the number of jobs currently assigned to a resource is less than the predicted number of jobs that a resource can consume, assign more jobs from unassigned job queue or from the most expensive machines based on job state and feasibility. Assign job to a resource only when there is enough budget available.
     - ii. Alternatively, if a resource has more jobs than it can complete by the deadline, move those extra jobs to unassigned job queue.
   - [DISPATCHER with Policy]
   - d. The dispatcher takes care of submission of jobs to remote machines with submission and resource policy and constraints depending on resource type (time or space shared).

Figure 20. DBC scheduling with cost optimization.

does not want to create a report, then he or she can pass '`null`' as the name of the ReportWriter entity. Note that users can choose any name for the ReportWriter entity and for the class name since all entities are identified by their name defined at the runtime.

4.2.2. Deadline and budget constrained cost optimization scheduling algorithm

We have simulated deadline and budget constrained (DBC) scheduling algorithms, cost optimization, time optimization, and none optimization, presented in [7]. We have also proposed a new scheduling algorithm, called cost–time optimization [22], which is a variant of cost and time optimization algorithms and simulated its performance.

The steps for implementing DBC cost optimization scheduling algorithms within economic broker simulator are shown in Figure 20. This algorithm attempts to process jobs as economically as possible within the deadline and budget. The results of the simulation are discussed in the next section.

4.2.3. Determining the deadline and budget

A D factor close to 1 signifies the user's willingness to set a highly relaxed deadline, which is sufficient to process an application even when only the slowest resources are available. Similarly a B factor close to 1 signifies that the user is willing to spend as much money as required, even when only the most expensive resource is used. The user jobs are scheduled on the Grid through its own broker. The broker uses these factors in determining the absolute deadline (see Equation (1)) and budget (see Equation (2)) values for a given execution scenario at runtime as follows.

Computing the deadline:

$$Deadline = T_{\text{MIN}} + D_{\text{FACTOR}} \times (T_{\text{MAX}} - T_{\text{MIN}}) \tag{1}$$

where T_{MIN} is the time required to process all the jobs, in parallel, giving the fastest resource the highest priority; T_{MAX} is the time required to process all the jobs, serially, using the slowest resource. An application with $D_{\text{FACTOR}} < 0$ would *never* be completed. An application with $D_{\text{FACTOR}} \geqslant 1$ would *always* be completed as long as some resources are available throughout the deadline.

Computing the budget:

$$Budget = C_{\text{MIN}} + B_{\text{FACTOR}} \times (C_{\text{MAX}} - C_{\text{MIN}}) \tag{2}$$

where C_{MIN} is the cost of processing all the jobs, in parallel within deadline, giving the cheapest resource the highest priority; C_{MAX} is the cost of processing all the jobs, in parallel within deadline, giving the most costly resource the highest priority. An application with $B_{\text{FACTOR}} < 0$ would *never* be completed. An application with $B_{\text{FACTOR}} \geqslant 1$ would *always* be completed as long as some resources are available throughout the deadline.

5. SCHEDULING SIMULATION EXPERIMENTS

To simulate application scheduling in GridSim environment using the economic Grid broker requires the modeling and creation of GridSim resources and applications that model jobs as Gridlets. In this section, we present resource and application modeling along with the results of experiments with quality of services driven application processing.

5.1. Resource modeling

We modeled and simulated a number of time- and space-shared resources with different characteristics, configurations, and capabilities from those in the WWG testbed. We have selected the latest CPU models AlphaServer ES40, Sun Netra 20, Intel VC820 (800EB MHz, Pentium III), and SGI Origin 3200 1X 500 MHz R14k released by their manufacturers Compaq, Sun, Intel, and SGI, respectively. The processing capability of these PEs in simulation time units is modeled after the base value of SPEC CPU (INT) 2000 benchmark ratings published in [23]. To enable the users to model their application processing requirements, we assumed the MIPS rating of the PEs to be the same as the SPEC rating.

Table II shows characteristics of resources simulated and their PE cost per time unit in G$ (Grid dollar). These simulated resources resemble the WWG testbed resources used in processing parameter

Table II. WWG testbed resources simulated using GridSim.

Resource name in simulation	Simulated resource characteristics: vendor, resource type, node OS, No. of PEs	Equivalent resource in WWG (hostname, location)	A PE SPEC/MIPS rating	Resource manager type	Price (G$/PE time unit)	MIPS per G$
R0	Compaq, AlphaServer, CPU, OSF1, 4	grendel.vpac.org, VPAC, Melbourne, Australia	515	Time-shared	8	64.37
R1	Sun, Ultra, Solaris, 4	hpc420.hpcc.jp, AIST, Tokyo, Japan	377	Time-shared	4	94.25
R2	Sun, Ultra, Solaris, 4	hpc420-1.hpcc.jp AIST, Tokyo, Japan	377	Time-shared	3	125.66
R3	Sun, Ultra, Solaris, 2	hpc420-2.hpcc.jp AIST, Tokyo, Japan	377	Time-shared	3	125.66
R4	Intel, Pentium/VC820, Linux, 2	barbera.cnuce.cnr.it, CNR, Pisa, Italy	380	Time-shared	2	190.0
R5	SGI, Origin 3200, IRIX, 6	onyx1.zib.de, ZIB, Berlin, Germany	410	Time-shared	5	82.0
R6	SGI, Origin 3200 IRIX, 16	Onyx3.zib.de, ZIB, Berlin, Germany	410	Time-shared	5	82.0
R7	SGI, Origin 3200, IRIX, 6	mat.ruk.cuni.cz, Charles U., Prague, Czech Republic	410	Space-shared	4	102.5
R8	Intel, Pentium/VC820, Linux, 2	marge.csm.port.ac.uk, Portsmouth, U.K.	380	Time-shared	1	380.0
R9	SGI, Origin 3200, IRIX, 4 (accessible)	green.cfs.ac.uk, Manchester, U.K.	410	Time-shared	6	68.33
R10	Sun, Ultra, Solaris, 8	pitcairn.mcs.anl.gov, ANL, Chicago, U.S.A.	377	Time-shared	3	125.66

sweep application using the Nimrod-G broker [24]. The PE cost in G$/unit time does not necessarily reflect the cost of processing when PEs have different capabilities. The brokers need to translate the cost into G$ per MI (million instructions) for each resource. Such translation helps in identifying the relative cost of resources for processing Gridlets on them.

5.2. Application modeling

We have modeled a task farming application that consists of 200 jobs. In GridSim, these jobs are packaged as Gridlets whose contents include the job length in MI, the size of the job input and output data in bytes, along with various other execution related parameters when they move between the broker and resources. The job length is expressed in terms of the time it takes to run on a standard resource PE with a SPEC/MIPS rating of 100. Gridlets processing time is expressed in such a way that

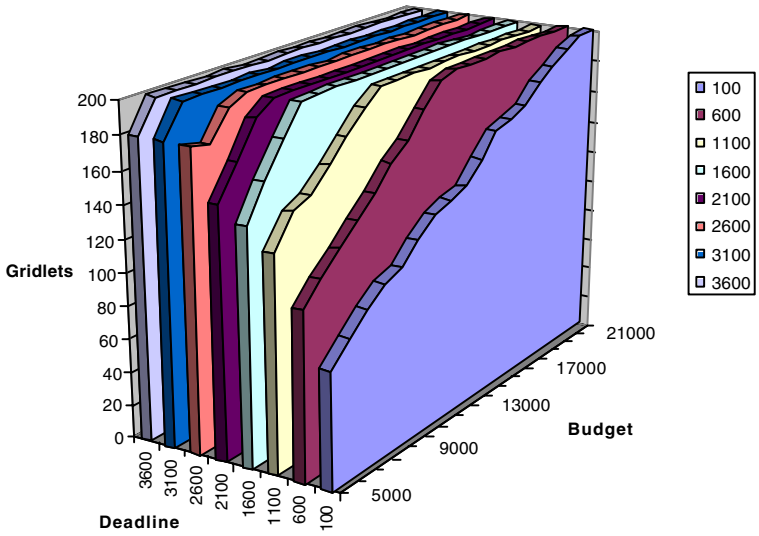


Figure 21. Number of Gridlets processed for different budget limits with a fixed deadline for each.

they are expected to take at least 100 time units with a random variation of 0–10% on the positive side of the standard resource. That means the Gridlet job length (processing requirements) can be at least 10 000 MI with a random variation of 0–10% on the positive side. This 0–10% random variation in the Gridlet job length is introduced to model heterogeneous tasks similar to those present in the real world parameter sweep applications.

5.3. DBC scheduling experiments with cost-optimization—for a single user

In this experiment, we performed scheduling experiments with different values of DBCs for a single user. The deadline is varied in simulation time from 100 to 3600 in steps of 500. The budget is varied from G$ 5000 to 22 000 in steps of 1000. For this scenario, we performed scheduling simulation for the DBC *cost-optimization* algorithm. The number of Gridlets processed, the deadline utilized, and the budget spent for different scheduling scenario is shown in Figures 21–24. From Figure 21, it can be observed that for a tight deadline (e.g. 100 time units), the number of Gridlets processed increases as the budget value increases. This is because, when a higher budget is available, the broker leases expensive resources to process more jobs within the deadline. Alternatively, when scheduling with a low budget value, the number of Gridlets processed increases as the deadline is relaxed (see Figure 22).

The impact of budget for different values of deadline is shown in Figure 23. In cost optimization scheduling, for a larger deadline value (see time utilization for deadline of 3600), the increase in budget value does not have much impact on resource selection. This trend can also be observed from the budget

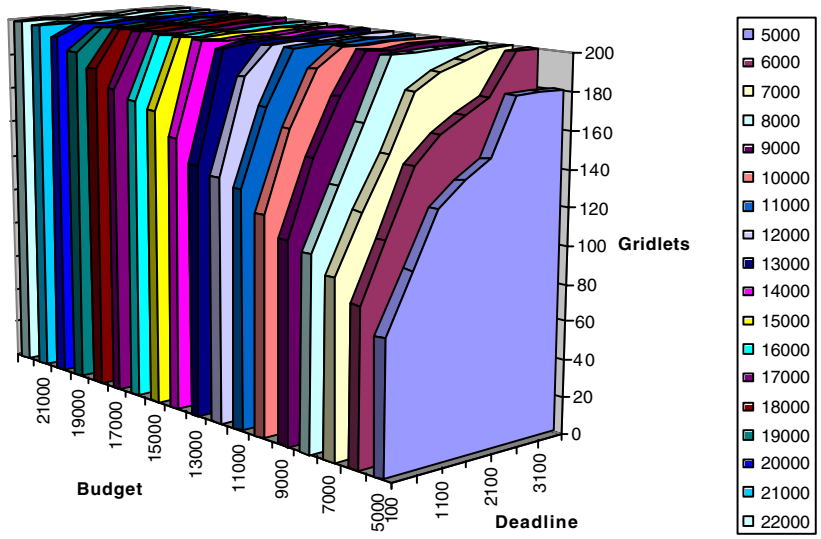


Figure 22. Number of Gridlets processed for different deadline limits with a fixed budget for each.

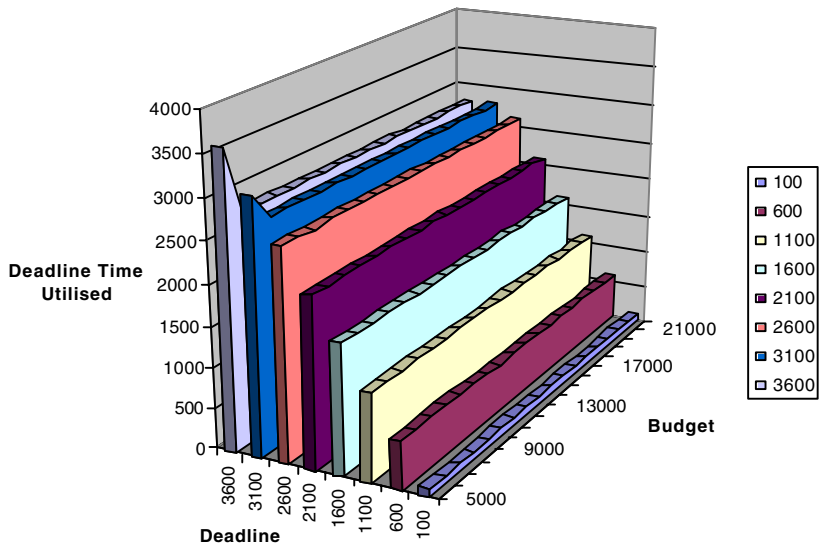


Figure 23. Deadline time utilized for processing Gridlets for different values of deadline and budget.

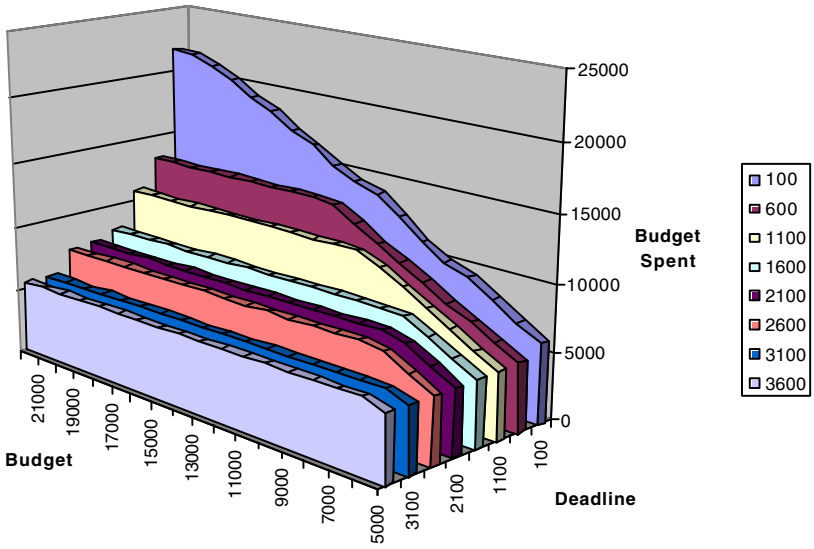


Figure 24. Budget spent for processing Gridlets for different values of deadline and budget.

spent for processing Gridlets with different deadline constraints (see Figure 24). When the deadline is too tight (e.g. 100), it is likely that the complete budget is spent for processing Gridlets within the deadline.

Figures 25–27 show the selection of resources for processing Gridlets for different budget values with a fixed deadline of 100, 1100, and 3100 (low, medium, and high deadline values), respectively. It can be observed that when the deadline is low, the economic broker also leases expensive resources to process Gridlets whenever the budget permits (see Figure 25). In this, all resources have been used depending on the budget availability. When the deadline is increased to a high value (a medium deadline of 1100), the broker processes as many Gridlets as possible on cheaper resources by the deadline (see Figure 26) and utilizes expensive resources if required. When the deadline is highly relaxed (a high deadline of 3100), the broker allocated Gridlets to the cheapest resource since it was able to process all Gridlets within this deadline (see Figure 27). In all three diagrams (Figures 25–27), the left most solid curve marked with the label 'All' in the resources axis represents the aggregation of all resources and shows the total number of Gridlets processed for the different budgets.

Let us now take a microscopic look at the allocation of resources at different times during the scheduling experimentation. Figures 28 and 30 show a trace of leasing resources at different times during the scheduling experiment for processing Gridlets for different budget values with a fixed deadline of 100 and 3100 (low and high deadline values), respectively. It can be observed that when the deadline value is low, the economic broker also leases expensive resources to process Gridlets whenever the budget permits. The broker had to allocate powerful resources even if they are expensive since the deadline is too tight (see Figure 28 for Gridlets completed and Figure 29 for budget spent in

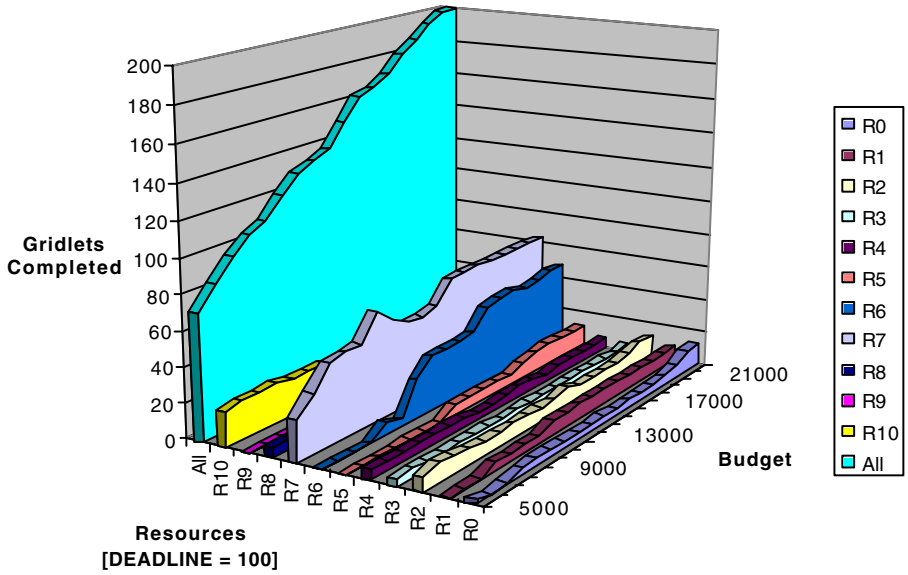


Figure 25. Gridlets processed on resources for different budget values with a low deadline.

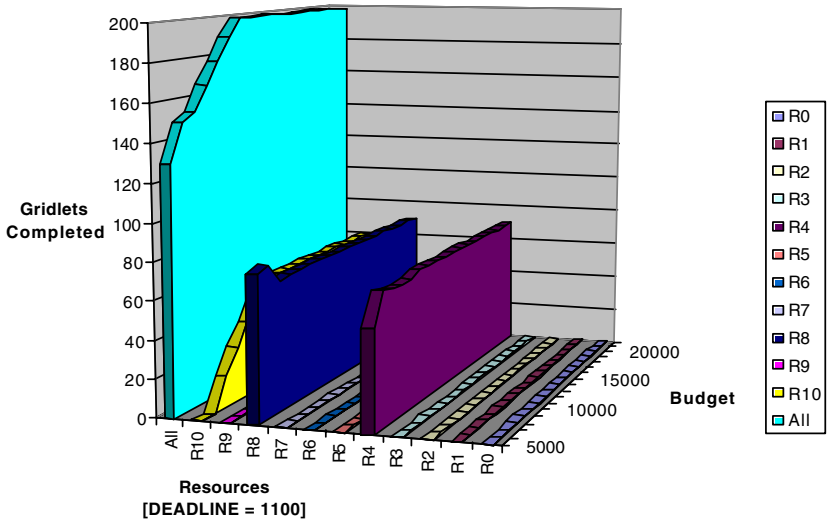


Figure 26. Gridlets processed on resources for different budget values with a medium deadline.

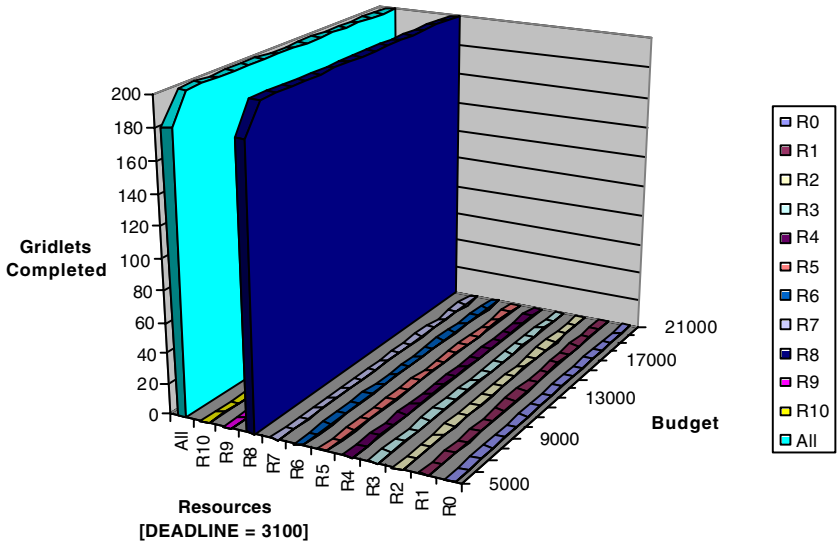


Figure 27. Gridlets processed on resources for different budget values with a high deadline.

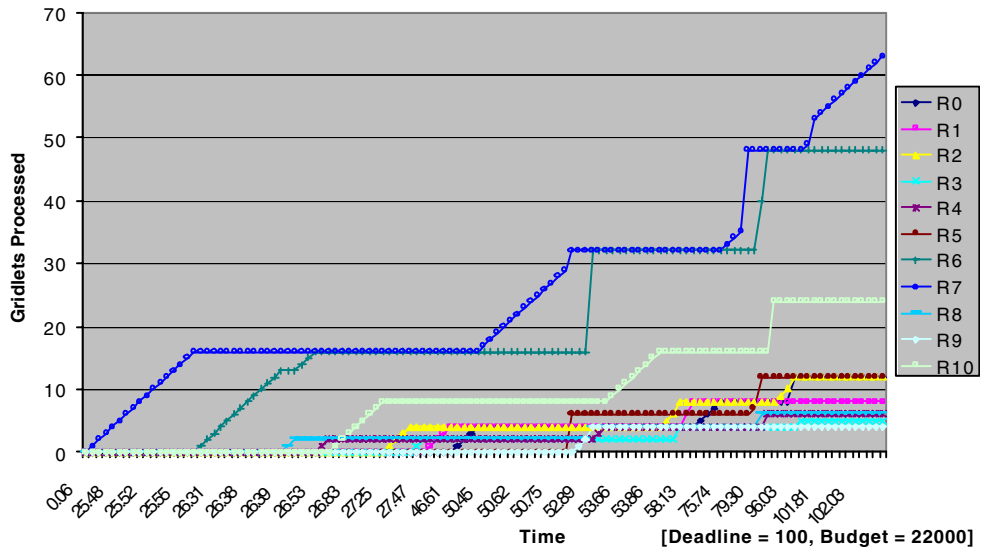


Figure 28. Trace of the number of Gridlets processed on resources for a low deadline and high budget constraints.

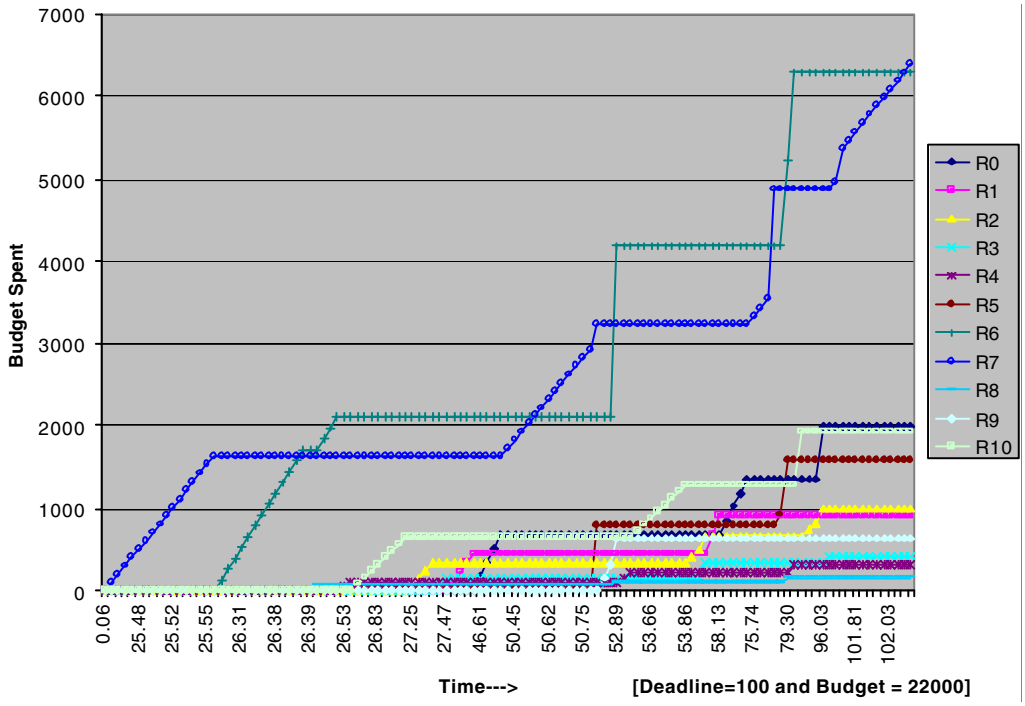


Figure 29. Trace of the budget spent for a low deadline and high budget constraints.

processing). But this is not the case when the deadline is highly relaxed (see Figure 30)—the broker leased just one resource, which happened to process all Gridlets within the given deadline. From the diagrams (Figures 28 and 29), it can be observed that the resource R7 has processed more Gridlets than the resource R6, but had to spent more budget on the resource R6 since it is more expensive than the resource R7.

A trace of the number of Gridlets committed to resources at different times depending on their performance, cost, and the user constraints (deadline and budget) and requirements (cost-optimization) is shown in Figures 31 and 32 for deadline values of 100 and 1100 time units, respectively. In both graphs it can be observed that the broker committed Gridlets to expensive resources only when required. The broker committed as many Gridlets as the cheaper resources could consume by the deadline. The remaining Gridlets were assigned to expensive resources. The broker used expensive resources in the beginning and continued to use cheaper resources until the end of the experiment. This ability of the economic Grid broker to select resources dynamically at runtime demonstrates its adaptive capability driven by the user's quality of service requirements.

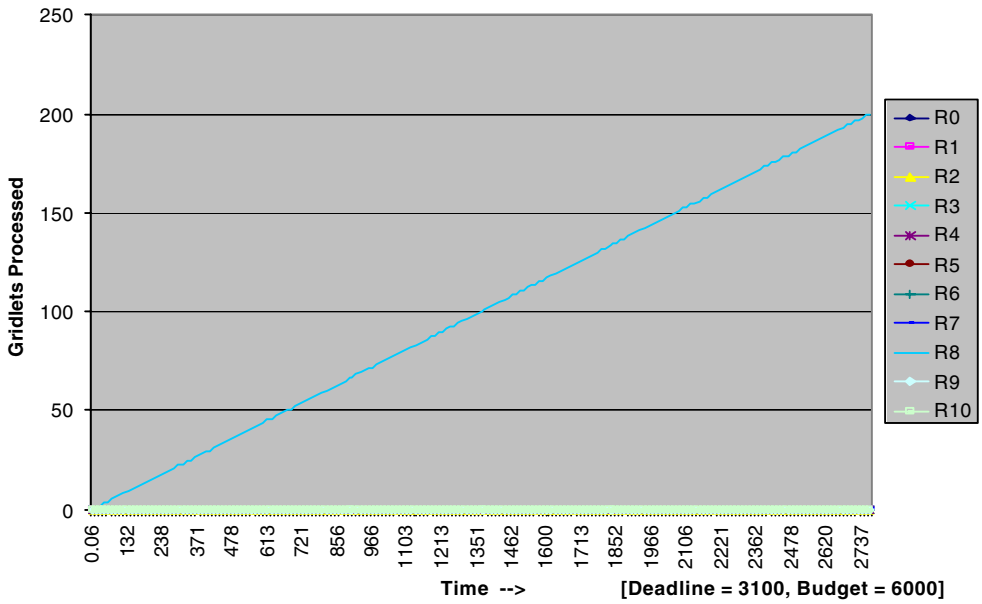


Figure 30. Trace of the number of Gridlets processed on resources for a high deadline and low budget constraints.

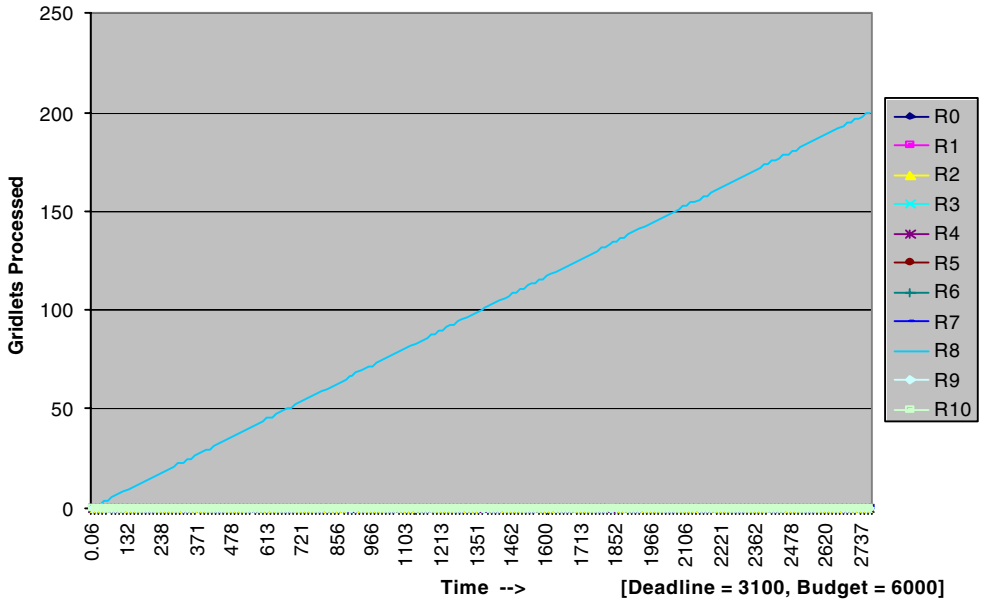


Figure 31. Trace of the number of Gridlets committed to resources for a low deadline and high budget constraints.

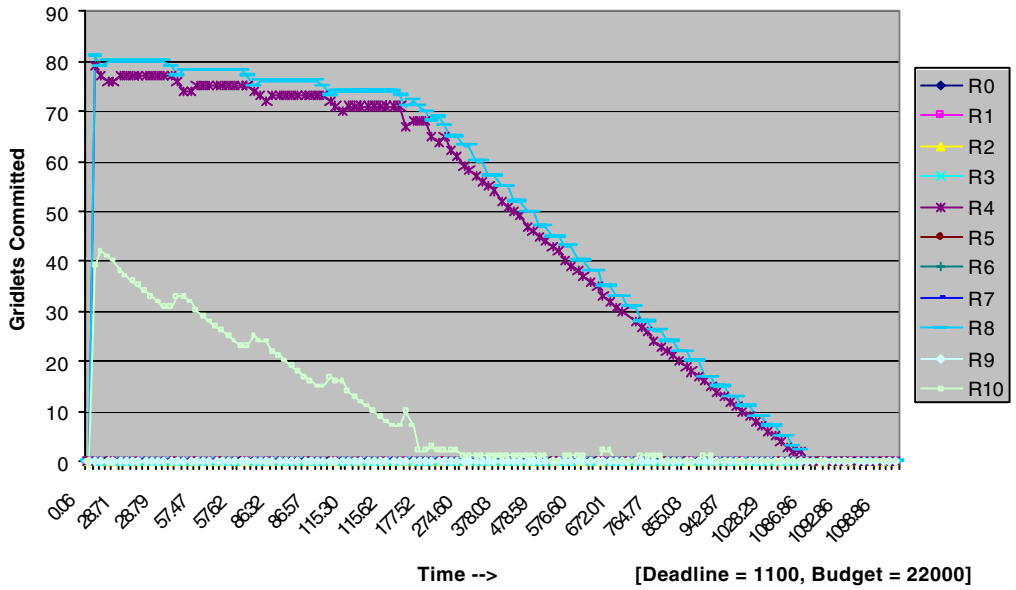


Figure 32. Trace of the number of Gridlets committed to resources for a medium deadline and high budget constraints.

5.4. DBC scheduling experiments with cost-optimization—for multiple users

In the second experiment, we explored distributed economic scheduling for a varying number of users competing for the same set of resources. All users are modeled to have similar requirements to enable comparison among them and understand the overall scenario. Each user application contains 200 Gridlets with small variation as explained in the application modeling section. We modeled a varying number of users in series from 1, 10, 20, and so on up to 100, each with their own broker scheduling Gridlets on simulated WWG testbed resources (listed in Table II). We explored the scheduling of Gridlets for different budget values varied from 5000 to 22 000 in steps of 1000. For this scenario, we performed two scheduling experiments with two different values of deadline for the DBC constrained *cost minimization* algorithm.

5.4.1. User deadline: 3100 time units

The number of Gridlets processed, the average time at which the simulation is stopped, and the budget spent for different scheduling scenarios for each user with a deadline constraint of 3100 time units is shown in Figures 33–35. From Figure 33, it can be observed that as the number of users competing for the same set of resources increases, the number of Gridlets processed for each user decreases because they have a tight deadline. Where there are few users (e.g. 1 or 10 users in this case), they are able

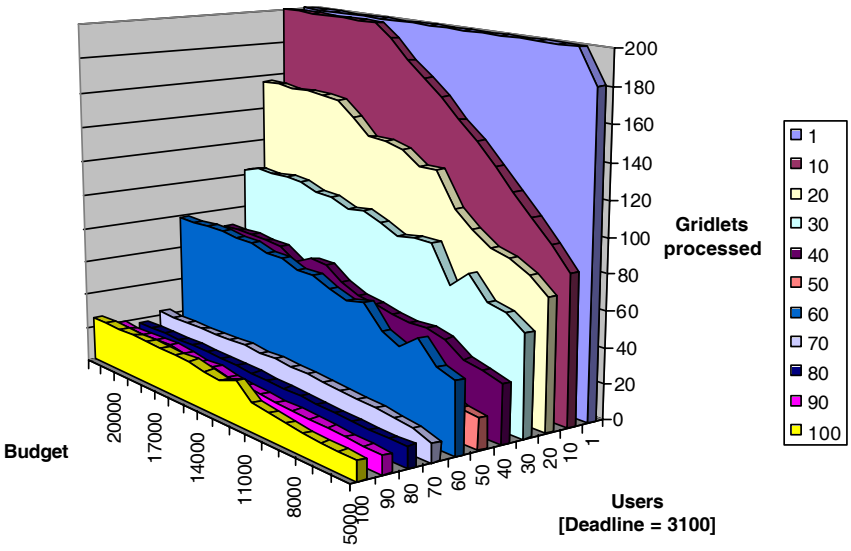


Figure 33. Number of Gridlets processed for each user when a varying number of users are competing for resources.

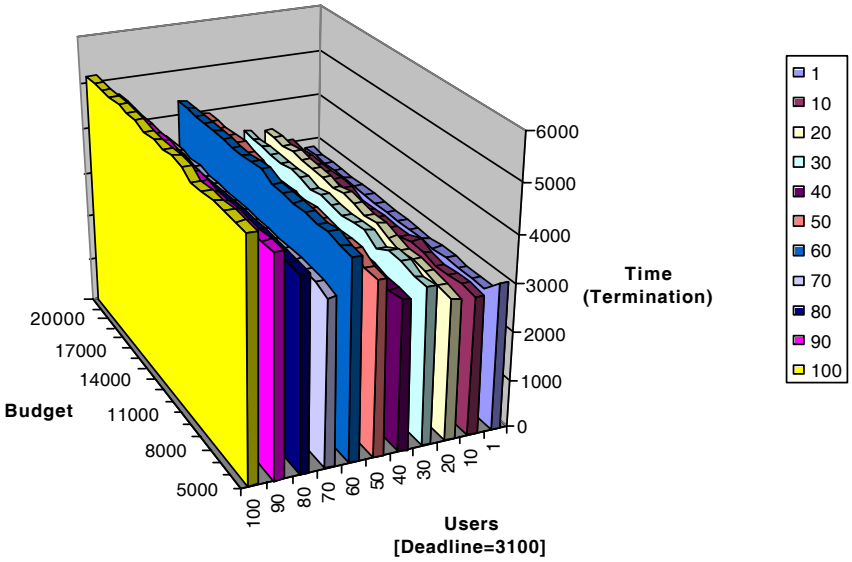


Figure 34. The average time at which the user experiment is terminated with a varying number of users competing for resources. When there are large numbers of users arriving at different times, they are likely to impact on the schedule and the execution time of jobs already deployed on resources. The broker waiting for the return of the jobs that are deployed on resources leads to the termination time exceeding the soft deadline unless the execution of jobs is cancelled immediately.

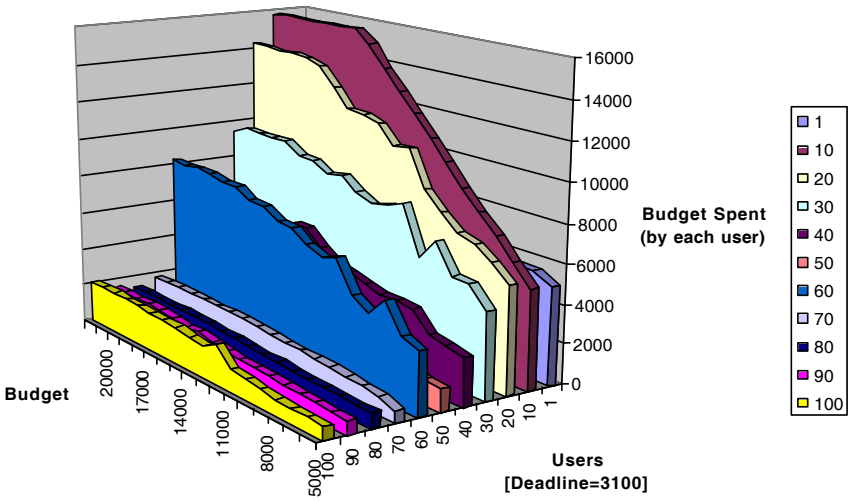


Figure 35. The average budget spent by each user for processing Gridlets.

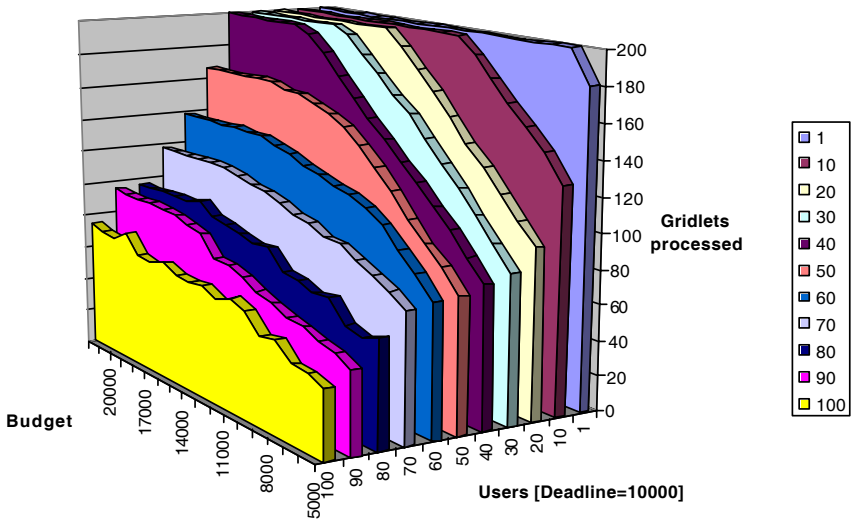


Figure 36. Number of Gridlets processed for each user when a varying number of users are competing for resources.

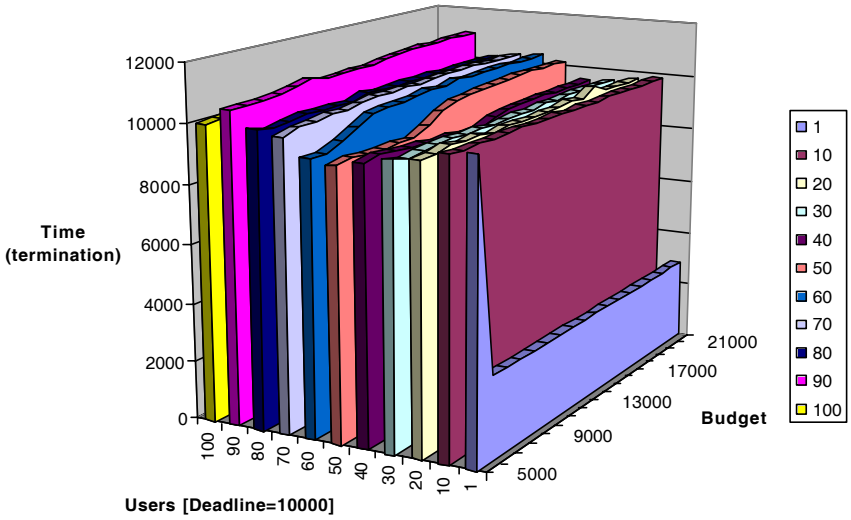


Figure 37. The average time at which the user experiment is terminated with a varying number of users competing for resources.

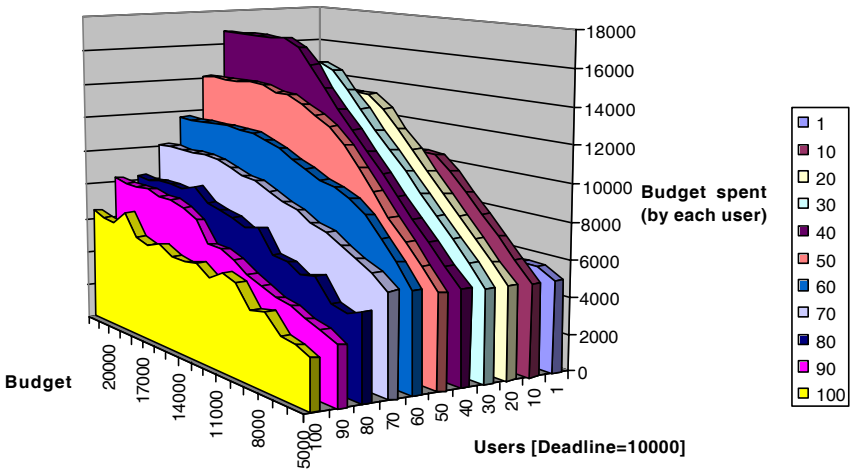


Figure 38. The average budget spent by each user for processing Gridlets.

to process all jobs in most cases when the budget is increased. Figure 34 shows the time at which the broker terminated the processing of Gridlets. When a large number of users are competing (e.g. 100) for resources, it can be observed that the broker exceeded the deadline. This is because the broker initially planned scheduling Gridlets for the period of the deadline, but that schedule had to be terminated because competing users had already occupied the high resource share well before the recalibration phase (the first establishment of the amount of resource share available to the user, which of course can change). Figure 35 shows the average budget spent by each user for processing the Gridlets shown in Figure 33, which is also clear from the graphic similarity between the two diagrams when a large number of users are competing for resources.

5.4.2. User deadline: 10 000 time units

The number of Gridlets processed, the average time at which the simulation is stopped, and the budget spent for different scheduling scenarios for each user with a deadline constraint of 10 000 time units is shown in Figures 36–38. In this experiment, the number of Gridlets processed for each user improved substantially due to the relaxed deadline constraint compared to the previous experiment (see Figures 33 and 36). As the number of users competing for resources increased, the number of Gridlets processed for each user decreased. However, when the budget was increased, the number of Gridlets processed increased. Unlike the previous experiment, the broker is able to learn and make better predictions on the availability of resource share and the number of Gridlets that can be finished before the deadline. As the deadline was sufficient enough to revisit the past scheduling decisions, the broker was able to ensure that the experiment is terminated within the deadline for most of the time (see Figure 37). The average budget spent by each user for processing Gridlets is shown in Figure 38, which is also clear from the graphic similarity between Figures 36 and 38 when a large number of users are competing for resources.

6. CONCLUSION AND FUTURE WORK

We discussed an object-oriented toolkit, called GridSim, for resource modeling and scheduling simulation. GridSim simulates time- and space-shared resources with different capabilities, time zones, and configurations. It supports different application models that can be mapped to resources for execution by developing simulated application schedulers. We have discussed the architecture and components of the GridSim toolkit along with steps involved in creating GridSim based application-scheduling simulators.

The implementation of the GridSim toolkit in Java is an important contribution since Java provides a rich set of tools that enhance programming productivity, application portability, and a scalable runtime environment. As the JVM is available for single, multiprocessor shared or distributed machines such as clusters, GridSim scales with them due to its concurrent implementation. Also, we were able to leverage the existing basic discrete-event infrastructure from SimJava while implementing the GridSim toolkit.

We have developed a Nimrod-G like economic Grid resource broker simulator using GridSim and evaluated a number of scheduling algorithms based on deadline and budget based constraints. This helped us in evaluating performance and scalability of our scheduling policies with different Grid configurations such as varying the number of resources, capability, cost, users, and processing

requirements. The results are promising and demonstrate the suitability of GridSim for developing simulators for scheduling in parallel and distributed systems.

A better network model is required to support the application model with tasks collaborating and exchanging partial results among themselves in a P2P fashion. Future work focuses on strengthening the network model by supporting various types of networks with different static and dynamic configurations and cost-based quality of services. The resource models need to be enhanced by interfacing with off-the-shelf storage I/O simulators. We have created a resource model for advance reservation and we will be implementing its scheduling simulation. To enable simulation of Grid resource management and scheduling with economic models such as tenders and auctions [24], we plan to integrate or support the FIPA (Foundation for Intelligent Physical Agents) standards [25] based interaction protocol infrastructure and extend the resource model to support them, along with quality of service guarantees. Efforts are currently underway, to develop and simulate an economic-based scheduler for single administrative domain resources such as clusters.

SOFTWARE AVAILABILITY

The GridSim toolkit software with source code can be downloaded from the project Web site: http://www.buyya.com/gridsim/.

ACKNOWLEDGEMENTS

We would like to thank David Abramson for his support and comments on improving the work. We thank John Crossley and Rob Gray for proofreading the paper. We thank Ajith Abraham and his family for their hospitality during Rajkumar's visit to the Gippsland campus. We thank Marcelo Pasin for his help in modeling resources with SPEC benchmark ratings.

REFERENCES

1. Foster I, Kesselman C (eds.). *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann: San Mateo, CA, 1999.
2. Oram A (ed.). *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly, 2001.
3. Baker M, Buyya R, Laforenza D. The Grid: International efforts in global computing. *Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, Rome, Italy, 31 July–6 August 2000.
4. Abramson D, Giddy J, Kotler L. High performance parametric modeling with Nimrod/G: Killer application for the global Grid? *Proceedings International Parallel and Distributed Processing Symposium (IPDPS 2000)*, Cancun, Mexico, 1–5 May 2000. IEEE Computer Society Press, 2000.
5. Buyya R, Abramson D, Giddy J. Nimrod/G: An architecture for a resource management and scheduling system in a global computational Grid. *Proceedings 4th International Conference and Exhibition on High Performance Computing in Asia-Pacific Region (HPC ASIA 2000)*, Beijing, China, 14–17 May 2000. IEEE Computer Society Press, 2000.
6. Buyya R, Abramson D, Giddy J. An economy driven resource management architecture for global computational power Grids. *Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*, Las Vegas, NV, 26–29 June 2000. CSREA Press, 2000.
7. Buyya R, Giddy J, Abramson D. An evaluation of economy-based resource trading and scheduling on computational power Grids for parameter sweep applications. *Proceedings of the 2nd International Workshop on Active Middleware Services (AMS 2000)*, Pittsburgh, PA, 1 August 2000. Kluwer Academic Press, 2000.

8. Buyya R, Stockinger H, Giddy J, Abramson D. Economic models for management of resources in peer-to-peer and Grid computing. *SPIE International Conference on Commercial Applications for High-Performance Computing*, Denver, CO, 20–24 August 2001.
9. Buyya R. The World-Wide Grid. http://www.csse.monash.edu.au/~rajkumar/ecogrid/wwg/.
10. Weissman J. Grids in the classroom. *IEEE Distributed Systems Online* 2000; **1**(3). http://www.computer.org/dsonline/archives/ds300/ds3eduprint.htm.
11. Buyya R (ed.). *High Performance Cluster Computing: Architectures and Systems*, vol. 1. Prentice-Hall: Englewood Cliffs, NJ, 1999.
12. CACI. *Simscript: A Simulation Language for Building Large-scale, Complex Simulation Models*. CACI Products Company: San Diego, CA. http://www.simscript.com/simscript.cfm.
13. Bagrodia R, Meyer R, Takai M, Chen Y, Zeng X, Martin J, Park B, Song H. Parsec: A parallel simulation environment for complex systems. *IEEE Computer* 1998; **31**(10).
14. Howell F, McNab R. SimJava: A discrete event simulation package for Java with applications in computer systems modelling. *Proceedings of the 1st International Conference on Web-based Modelling and Simulation*, San Diego, CA. Society for Computer Simulation, 1998.
15. Varga A. The OMNeT++ discrete event simulation system. *Proceedings of the European Simulation Multiconference (ESM 2001)*, Prague, Czech Republic, 6–9 June 2001.
16. Aida K, Takefusa A, Nakada H, Matsuoka S, Sekiguchi S, Nagashima U. Performance evaluation model for scheduling in a global computing system. *The International Journal of High Performance Computing Applications* 2000; **14**(3).
17. Song H, Liu X, Jakobsen D, Bhagwan R, Zhang X, Taura K, Chien A. The MicroGrid: A scientific tool for modeling computational Grids. *Proceedings of IEEE Supercomputing (SC 2000)*, Dallas, TX, 4–10 November 2000.
18. Casanova H. Simgrid: A toolkit for the simulation of application scheduling. *Proceedings 1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001)*, Brisbane, Australia, 15–18 May. IEEE Computer Society Press, 2001.
19. Foster I, Kesselman C. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications* 1997; **11**(2):115–128.
20. Aridor Y, Factor M, Teperman A. cJVM: A single system image of a JVM on a cluster. *Proceedings 29th International Conference on Parallel Processing (ICPP 99)*, Fukushima, Japan, September 1999. IEEE Computer Society Press, 1999.
21. Silva L, Buyya R. Parallel programming paradigms. *High Performance Cluster Computing: Programming and Applications*, vol. 2, ch. 2. Prentice-Hall: Englewood Cliffs, NJ, 1998.
22. Buyya R, Murshed M. A deadline and budget constrained cost–time optimize algorithm for scheduling parameter sweep applications on the Grid. *GridSim Toolkit Release Document*, December 2001. http://www.buyya.com/gridsim.
23. SPEC. SPEC CPU2000 Results. http://www.specbench.org/osg/cpu2000/results/cpu2000.html [30 January 2002].
24. Buyya R, Abramson D, Giddy J, Stockinger H. Economic models for resource management and scheduling in Grid computing. *Concurrency and Computation: Practice and Experience* 2002; **14**(13–15):1507–1542. http://www.buyya.com/papers/emodelsgrid.pdf.
25. Foundation for Intelligent Physical Agents (FIPA). Interaction and negotiation protocols. http://www.fipa.org/.

